Claro como cristal

A Metodologia humana para pequenas equipas,

Incluindo

As sete propriedades de projetos de software eficazes

Alistair Cockburn
Humans and Technology

última versão: 17 junho de 2004

Traduzido em outubro 2019 por Frederico Rodrigues
www.fredericrodrigues.fr

Prefácio

Crystal Clear: Algumas regras fundamentais para manter um pequeno projeto na zona de segurança.

Tem apenas suficiente recursos para entregar o sistema. Não quer que a equipas escrevam documentos longos, mas estão esquecendo as coisas que deveriam saber. Não gosta de processos de desenvolvimento de software muito formais, mas quer que sua equipas trabalhem melhor e não aleatoriamente. Deseja particularmente que o software seja entregue com sucesso.

Considera sentar-se e escrever as discussões básicas que as equipas devem ter, os produtos que merecem ser realizados com cuidado. Perguntou a si mesmo:

> **O que outras, pequenas equipas de projetos bem-sucedidas fazem? Quais são as práticas que eles usam?**

Este livro responde a essas perguntas. É o resultado de dez anos de "debriefing" de pequenas equipas de sucesso. A maioria delas repetiu o mesmo método:

- **Juntar as pessoas na mesma sala, comunicando com frequência e com boa vontade;**
- **Pôr a maior parte da burocracia de lado e permitir à equipa de se projetar;**
- **Ter um verdadeiro utilizador envolvido;**
- **Ter um bom conjunto de testes de regressão automatizados disponíveis;**
- **Produzir funcionalidade entregável cedo e com frequência.**

Faça tudo isso, e a maioria dos detalhes do processo vão se desenrolar por si. Este livro apresenta uma das metodologias mais eficientes e habitáveis que você pode encontrar, Crystal Clear. É uma metodologia "*humano centrada*", mais simplesmente descrita como o seguinte:

> **O designer-chefe e 2-7 outros desenvolvedores em uma grande sala ou salas adjacentes, com quadros de informação, tais como quadros e cartazes na parede, tendo acesso a utilizadores-chave, as distrações são mantidas longe, entregando software funcional, testado, utilizável cada mês ou dois (OK, três do ponte de vista exterior), refletindo periodicamente e ajustando o seu estilo de trabalho.**

Esta simples recomendação se baseia em experiência e teoria. O desenvolvimento de software pode ser caracterizado como um *jogo cooperativo* constrangido economicamente

de invenção e comunicação 1. A forma como a equipa joga cada jogo tem tudo a ver com o resultado do projeto e do software resultante. Crystal Clear aborda o jogo económico-cooperativo diretamente, abordando aonde prestar atenção, aonde simplificar, e como variar as regras. Um certo número de equipas compartilharam comigo - e agora com você, através deste livro - exemplos de suas regras, produtos de trabalho e até design de organização.

Muitas das "melhores" metodologias foram rejeitadas por equipas como sendo demasiado restritivas, muito invasivas, ou muito difíceis de apreender. O Crystal Clear não aspira ser a "melhor" metodologia; aspira ser a metodologia "suficiente", afim que a sua equipa vai moldá-lo para si e, em seguida, realmente usá-lo.

Origem do material deste livro

O grupo IBM Consulting pediu-me em 1991 para escrever uma metodologia sobre projetos de tecnologia orientada a objetos. Não sabendo o suficiente sobre metodologias naquele momento para tomar as decisões cruciais, e por sugestão de meu chefe, Kathy Ulisse 2, comecei a entrevistar as equipas de projeto. O que me disseram era muito diferente do que eu estava lendo nos livros. Em particular, eles enfatizaram aspectos não abordados nos artigos de metodologia: comunicação permanente, a moral, o acesso a utilizadores finais, e assim por diante. Não demorou muito para que estas questões estejam em forte contraste com os projetos bem-sucedidos em relação aos fracassados. Eu vim para ver estas questões, e não as técnicas de design, como a chave para alcançar um resultado de projeto bem-sucedido.

Eu tenho que tentar essas ideias como consultor líder em projeto de $ 15 milhões, a preço fixo, com equipa de 45 pessoas. As ideias funcionaram como anunciado (juntamente com muita criatividade ao longo do caminho), e mostrou-se como fatores de sucesso. Redigi as lições aprendidas a partir das entrevistas do projeto e esse projeto em *Sobrevivendo* a *Projetos orientados a objetos* 3.

Um trio em particular mostrou-se repetidamente: posicionamento da equipa, entrega frequente, e acesso a um utilizador experiente. As diferenças de resultados entre os projetos que fizeram isso e não excedeu em muito qualquer outra lista de práticas. Este livro constrói o método a partir desse trio.

Os projetos em minha carreira têm sido geralmente de preço fixo, com perímetro fixo. As pessoas costumam subestimar esses projetos, o que significa que a única maneira

1 Descrito em detalhe no *Desenvolvimento Ágil de Software (* Cockburn 2002) e repetido na primeira resposta do *Questioned.*

2 Obrigado pelo conselho brilhante, Kathy!

3 Os debriefings de projeto foram a base para a minha tese de doutorado, "Pessoas e Metodologias em Desenvolvimento de Software" (Cockburn, 2003).

de a equipas entregar a tempo é de ser muito criativo com o seu processo de desenvolvimento. Ao contrário da maioria dos outros autores do Manifesto Ágil, fui ter com os princípios ágeis através da necessidade de eficiência, não a necessidade de lidar com requisitos que mudam rapidamente.

Como resultado, a Crystal Clear é bem adequado para o contexto de preço fixo. Se você está em tal situação, utilize o planificação, comunicação e mecanismos de informação que descrevem a cumprir o seu prazo (provavelmente irrealista), e só tome cuidado para não alterar os requisitos no início de cada iteração. Se
você tiver um projeto exploratório em que os requisitos são desconhecidos ou mudam, não há problema: em seguida, *permita* que os requisitos mudem no início de cada iteração.

Crystal Clear na Família cristal

Crystal é uma família de metodologias com um código genético comum, que enfatiza a entrega frequente, estreita comunicação e melhoria reflexiva. Não há *1* metodologia de cristal. Existem metodologias de cristal diferentes para diferentes tipos de projetos. Cada projeto ou organização utiliza o código genético para gerar novos membros da família.

O nome "cristal" vem da minha caracterização de projetos ao longo de duas dimensões, tamanho e criticidade, correspondentes à de minerais, cor e dureza (ver Figura 7-1).

Projetos maiores, que exigem mais coordenação e comunicação, correspondem a
***cores* mais escuras (transparente, amarelo, laranja, vermelho, e assim por diante). Projetos de sistemas que podem causar mais danos, necessidade de mais robusteza na metodologia, mais regras de validação e verificação. A metodologia de quartzo é adequada para alguns desenvolvedores que criam um sistema de faturação. A mesma equipa que controla o movimento das hastes de boro em um reator nuclear precisa de uma metodologia de diamante, uma chamada para verificações repetidas, tanto na concepção com na implementação de seus algoritmos.**

Caracterizo metodologias de cristal por cor, de acordo com o número de pessoas que estão sendo coordenadas: Transparente é para equipas dedicadas de oito ou menos, Amarelo é para equipas de 10-20 pessoas, Laranja é para 20-50 pessoas, vermelho é para 50-100 pessoas, e assim por diante, através de castanho, azul e violeta. Crystal Laranja é
descrito em *Sobrevivendo a Projetos Orientados a Objetos,* e sua variante Crystal Orange / Web é descrito em *Desenvolvimento Ágil de Software.* Acho que, exceto em projetos críticos da vida, as pessoas podem participar nas atividades de verificação através da formação à metodologia e oficinas de melhoramento continuo 4.

---

4 Se a sua empresa desenvolve FDA ou outros sistemas críticos, pode querer configurar três metodologias de base, um para um projeto transparente (quartzo), um para amarelo ou

o código genético de cristal é feito de:

- **um modelo de jogo económico-cooperativo,**
- **seleção de *prioridades,***
- **seleção de *propriedades,***
- **seleção de *princípios,***
- **seleção de *técnicas,* e**
- **exemplos de projeto.**

O modelo de jogo económico-cooperativo diz que o desenvolvimento de software é uma série de "jogos" cujos movimentos consistem em nada mais além de inventar e comunicar, que normalmente é de recursos limitados. Cada jogo da série tem dois objetivos que competem por recursos: para entregar o software neste jogo, e criar o próximo jogo da série. O jogo nunca se repete, de modo que cada projeto exige estratégias ligeiramente diferentes de todos os jogos anteriores. O modelo de jogo económico-cooperativo leva as pessoas em um projeto para pensar sobre o seu trabalho de uma forma muito específica, focada e eficaz.

As prioridades comuns para a família de cristal são

- ***segurança* no resultado do projeto,**
- ***eficiência* no desenvolvimento, e**
- ***aceitabilidade* das convenções (os desenvolvedores podem viver com elas). Cristal tem a**

equipas boi projeto para sete propriedades de segurança, as três primeiras propriedades dos quais são fundamentais para Cristal. Os outros podem ser adicionados em qualquer ordem para aumentar a margem de segurança. As propriedades são

- ***entrega frequente,***
- ***estreita comunicação***
- ***melhoria reflexiva,***
- **segurança pessoal (o primeiro passo de confiança),**
- **foco,**
- **fácil acesso a utilizadores experientes, e**
- **ambiente técnico com testes automatizados, gestão de configuração e integração frequente.**

princípios de cristal são descritos em detalhes no *Desenvolvimento Ágil de Software* (Cockburn 2002). Entre elas estão algumas ideias centrais:

- **A quantidade de detalhes necessários nos requisitos, documentos de projeto e planificação varia com as circunstâncias do projeto, especificamente**
  - **a extensão dos danos que pode ser causada por defeitos detetados e**

Orange, e um terceiro para todos os sistemas validados. Aqueles três devem podem fornecer base adequada para moldar a qualquer um dos projetos na sua empresa.

- **a frequência de colaboração pessoal apreciado pelas equipas.**

- **Pode não ser possível eliminar *todos os* produtos intermediários de trabalho e notas promissórias, como requisitos, documentos de projeto e planos de projeto, mas eles podem ser reduzidos na medida em que**
  - **caminhos de comunicação informais curtas e pragmáticos estão disponíveis para as equipas;**
  - **é entregue software funcional e testado mais cedo e com frequência.**
- **A equipa ajusta continuamente suas convenções de trabalho para averiguar**
  - **as personalidades particulares nas equipas,**
  - **o atual ambiente de trabalho local, e**
  - **as peculiaridades da tarefa específica.**

Entre o resto são curvas de "trade-off" que destacam as implicações de custo de diferentes mecanismos de comunicação, diferentes situações de projeto e estratégias diferentes para desenvolvimento simultâneo. Eu usei esses princípios para criar Crystal Clear, mas não os considere separadamente neste livro.

O pacote de cristal inclui técnicas de amostra selecionada, inclusive para formação metodologia, planificação e melhoria reflexiva. O cristal não requer nenhuma técnica específica a ser utilizada por qualquer das pessoas no projeto, para que estas técnicas estejam incluídas apenas como um conjunto inicial.

Cada membro da família de Cristal é gerado no início de um projeto de formação de uma metodologia de base de acordo com o código genético. Dado que a situação muda ao longo do tempo, a metodologia é afinada novamente durante o curso do projeto. Ambos a escultura e afinação são realizadas com rapidez suficiente que o tempo gasto fica reembolsado dentro do prazo do projeto.

Crystal Clear é uma otimização de cristal que pode ser aplicada quando a equipa é composta de dois a oito pessoas dentro da mesma sala ou salas adjacentes. A propriedade de estreita comunicação é reforçada com a comunicação "osmótica", o que significa que as pessoas ouvem uns aos outros a discutir as prioridades do projeto, status, requisitos e projeto em uma base diária. Este reforço da comunicação permite que a equipas trabalhem mais a partir da comunicação tácita e pequenas notas do que de outra forma seria impossível.

Porque cada empresa e projeto é um pouco diferente, mesmo Crystal Clear não está completamente especificado. O primeiro passo na adoção de Crystal Clear é descobrir pontos fracos e fortes da sua organização, e para atender as recomendações do Crystal Clear em torno deles para capitalizar sobre os pontos fortes e cobrir as fraquezas.

Para algumas organizações, isso é muito trabalho. Crystal Clear não é para essas organizações. É por grupos que querem construir a sua própria maneira, pessoal, forte e eficaz para entregar software repetidamente.

Crystal Clear compartilha algumas características com o XP, mas geralmente é menos exigente. Você pode pensar nisso como uma alternativa mais descontraído, um lugar para cair de volta para se XP não está funcionando para o grupo, ou um trampolim para obter algumas práticas ágeis no lugar antes de saltar para XP.

Este livro da série Desenvolvimento Ágil

Este livro faz parte da *Desenvolvimento Ágil de Software* série editada por Jim Highsmith e eu. A série descreve a teoria e a prática de desenvolvimento de software.

Os fundamentos teóricos são discutidos em quatro livros: *Desenvolvimento Ágil de Software (* Cockburn 2002), *Adaptive Desenvolvimento de Software (* Highsmith 2001), *Agile Project Management (* Highsmith 2004), e *Magra (Software Development* Poppendieck 2003).

Os outros livros da série pegar o fio, quer por descrever uma técnica para um determinado indivíduo ou função, uma técnica definida para toda a equipas, ou uma amostra de metodologia.

- **Encontramos a atenção para o papel de gerente de projeto no *Projetos sobreviver Object-Oriented (* Cockburn 1998), *Agile Project Management (* Highsmith 2004), e ao papel de requisitos escritor *Effective Use Cases escrito (* Cockburn 2001), *Padrões para Effective Use Cases (* Adolph 2002).**
- **Encontramos atenção para toda a equipas em *Melhorar as organizações de software* (Mathiassen 2001) e *Gestão de configurações (* Haas 2003).**
- **metodologias ágeis específicos são descritos em *DSDM (* Stapleton 2003), *Ágeis de Desenvolvimento de Software (Ecossistemas* Highsmith 2003), *Desenvolvimento iterativo e incremental: Guia de um Manager (* Larman 2004) e este livro. livros futuros seguirão o tema, acrescentando técnicas de colaboração e de saúde equipas.**

Como Ler este livro

Algumas pessoas vão ler este livro como uma introdução às técnicas de desenvolvimento ágil, e ser relativamente novo para emparelhar programação, desenvolvimento orientado a testes, comunicação osmótica, jogo económico-cooperativo, integração contínua, e radiadores de informação.

Se isso é você, então leia este livro praticamente direto, porque você é a pessoa para quem eu projetei a ordenação capítulo.

- **Eu escrevi o primeiro capítulo, *Explicado,* como uma troca de e-mail entre Cristal e me expor como essas equipas olhar para um estranho (lembre-se, era uma vez tudo novo para mim, também).**

- ***Aplicada (Propriedades)* ainda é o capítulo mais importante. Ele descreve o que a equipas está buscando, não o procedimento que usa para chegar lá.**
- ***Na prática (Estratégias e Técnicas)* deve dar-lhe uma alça sobre como alguns dos isso é feito.**

- ***(Processos) exploradas* descreve a cíclica núcleo processos de desenvolvimento a todas as metodologias ágeis (na verdade todos modernos). É algo que todos devem ser fluentes em.**

- **Apenas digitalizar o *produtos de trabalho* na primeira passagem, uma vez que é muito detalhado. Use-o como uma enciclopédia de amostras de produtos de trabalho quando você chegar tão longe.**
- ***Incompreendidos (erros comuns)* e *Questionado (frequently asked)* deve responder a perguntas sobre o que conta variações como ok e não-aprovado.**
- ***Testado (Um Estudo de Caso)* dá-lhe outra chance de ver a metodologia do lado de fora, uma vez que foi escrito para pessoas não familiarizadas com Crystal Clear. Ele também contém uma análise e um recomendações 9001 do auditor ISO, o qual emite luz a partir de um ângulo diferente.**

- **Ler *destilado* para ver se faz sentido nesse ponto. Se isso não acontecer, você pode ter que voltar para *questionado* para ver por que uma recomendação tão simples funciona. Algumas pessoas estão plenamente versado em desenvolvimento ágil moderna, incluindo design orientado a testes e integração contínua. Se você é uma dessas, eu sugiro que você vá diretamente para o capítulo *Destilada.* Depois disso (quando terminar rindo), leia *propriedades,***

provavelmente o mais importante capítulo no livro. Meu palpite é que você vai encontrar algumas ideias novas para experimentar em *Estratégias e técnicas.* Depois de olhar através desses capítulos, voltar a *destilado* para ver se tudo se encaixa para você.

Se você está dando isso para seu chefe ou gerente, a minha esperança é que o primeiro capítulo com os e-mails é o tipo de coisa que pode ser lido no avião ou na banheira. Um gerente ou executivo também deve ler a *(Processos) exploradas* capítulo, porque aprender como encaixar um processo cíclico na organização é importante.

designers de processo ou metodologia são bastante propensos a recorrer diretamente ao *Examinados (Produtos de Trabalho),* porque esta é uma maneira padrão de avaliação de metodologias (embora no caso de Crystal Clear, bastante insuficiente). Para completar a avaliação, porém, você precisa também de ler *Questionada (comparação)* para ver a comparação com outros sistemas de metodologia.

Finalmente, você estará pronto para começar. Neste ponto, leia a resposta à última *Pergunta, questão, "* Como faço para começar?" E trabalhar a partir daí. Isso o levará de volta para as técnicas de metodologia de modelagem e de reflexão e *Propriedades.*

Eu escrevi cada capítulo em seu próprio estilo e tom. Há uma razão para isto. Pessoas aprendendo uma metodologia estão em muito a mesma situação que os cegos tentando adivinhar a forma de um elefante, cada um sentindo uma parte diferente e chegando com uma resposta diferente. Cada pessoa. vindo de seu fundo original percebe coisas diferentes, e olha para coisas diferentes. Portanto, os nove capítulos diferentes são escritas de maneiras bem diferentes. Eu não espero que todos sejam felizes com cada capítulo, mas eu espero que todo mundo acha *alguns* capítulo que aborda seu e seu fundo e interesses individuais.

Reconhecimentos

Sou grato a muitas mais pessoas para este livro do que para qualquer um dos meus anteriores: pessoas que me contaram suas histórias, pessoas que tentaram as ideias, pessoas que contribuíram amostras, as pessoas que revisaram o texto, e as pessoas que me apoiaram emocionalmente.

A maioria das pessoas que me contaram suas histórias de projeto durante os últimos dez anos, não sei quanto valor eles fornecido como eles explicaram - até Pediu desculpa para!
- suas formas de trabalhar. Eles muitas vezes disse, "nós não usamos uma metodologia aqui, nós apenas . . ."Ou" Sinto muito, não se preocupam em fazer [xyz], ou manter os nossos documentos, mas temos encontrado. . ." Foi apenas comparando os resultados através de um número de projetos que eu descobri que em muitos desses casos, não queira nos foi sempre necessária, que havia uma força no caminho eles trabalharam.

Certas pessoas foram pioneiros em tentar essas ideias para fora em seus projetos. Jens Coldewey foi o primeiro, em 1998, Robert Volker em 1999, Gery Derbier em 2002, Stephen Sykes em 2003. Dr. Christopher Jones experimentou uma versão inicial em seus inocentes estudantes sênior de engenharia de software (que usou toda a tecnologia de implementação que se possa imaginar). Seus alunos mostrou-me aonde minha escrita era ambígua ou mal descrito. Stephen permissão tem para mim para incluir tanto o seu relatório de campo e análise do auditor neste livro, uma enorme contribuição.

Pete McBreen manteve lembrando-me que *pessoas* também são valiosos transições de um projeto para o próximo (algo que eu sabia, mas que manteve escapando das minhas descrições até Pete me lembrou novamente). Jeff Patton e Andy Pols eram parceiros de discussão contínua sobre os temas.

As seguintes pessoas contribuíram fotos de escritório e amostras de trabalho (Agradeço a cada novamente ao lado de suas amostras): Lise Hvatum, Jeff Patton, Ron Jeffries, Jonathan Casa, Nate Jones, Kay Johanssen, Darin Cummins, Randy Stafford, Mike Cohn, ... ?? ?

Luke Hohmann e Tom Poppendieck li tudo o que escreveu com rigor surpreendente e pegou erros de omissão, comissão e até mesmo intenção. Tom

surgiu com a idéia de datar os e-mails em vez de numeração deles (Duh, por que eu não tinha pensado nisso ?! Obrigado, Tom).

Os padrões Grupo Vale do Silício e, especialmente, Russ Rufer e Chris Lopez ler o manuscrito cuidadosamente não apenas uma vez, mas duas vezes, dando os seus comentários pensativo habituais e empurrando para trás em mim quando senti que eu tinha ido fora da pista. Russ argumentou me implacavelmente em algumas das seções até que eu finalmente o seu ponto.

Algumas pessoas lê-lo a partir da web e escreveu com correções e melhorias. Obrigado, Marco Cova, Todd Little, Alan Griffiths, Howard Medo, Victoria Einarsson, Paul Chisholm, Pierce McMartin, Phillipe Back, Todd Jonker, Chris Matts, Gain Wong, Jeremy Brown, John Rusk, Johannes Brodwall, ???

As pessoas da mesa redonda Salt Lake Agile Grupo mesmo "design da caixa" para Crystal um dia. As citações de topo daquele dia foram,

> **"Self-tuning, auto-correção, basta adicionar as pessoas!" "Funciona**
>
> **com biscoitos powdermilk!"**

eo testamonial vencedora:

> **"Eu usei cristal toda a minha vida, e eu nunca fui o mesmo!"**

Graças a Jim Highsmith, co-editor e parceiro no crime, para as muitas discussões esclarecedoras que moldaram as nossas ideias, a nossa série de livros e da maravilhosa Conferência de Desenvolvimento Agile em 2003.

Graças ao meu novo café favorito, o Coffee Break Salt Lake, que fica aberto até 02:00, tem clientela interessante, tomadas de energia que funcionam, e baklava e café turco para aqueles momentos raros. Graças a minha família: Deanna para o check-in com me no café às 1:00 da manhã para ver como a digitação ia (e depois me deixar dormir na parte da manhã), e Kieran, Sean e Cameron para sua consideração positiva de minha escrita hábito.

Índice

Capítulo 1

***explicou (* Vista do lado de fora)**

I destilada Crystal Clear, pedindo pequenas equipas de sucesso que eles iriam manter ou alterar nos caminhos que eles trabalhavam. O respondeu com esta aparentemente simples conjunto de regras. Este capítulo é escrito como uma troca de e-mail entre o cristal fictícios e me. Os e-mails que você encontrar as regras do lado de fora, como eu fiz em primeiro lugar, e permitam-me para empurrar para trás contra os relatórios de cristal, fazendo perguntas.

Prefácio para os e-mails: **Os seguintes e-mails são escritos para simular o que é como encontrar pela primeira vez uma equipas fazendo Crystal Clear. Ao escrever, eu descobri que era melhor para escrever como se cristal eram o único a fazer as entrevistas do projeto e ensinando Alistair-the-autor, embora na vida real Alistair fez a entrevista. No que se segue, Alistair chega a ser o malandrinho, com Crystal puxando-o relutantemente junto, mesmo repreendê-lo na ocasião.**

Aqui está o layout do escritório de cristal refere-se a:

Figura 1-1. Um ambiente de trabalho que mostra uma comunicação osmótica. (Graças a Tomax)

1 de Junho: **Caro Alistair,**

Apenas voltar de minha viagem a criogênica Comércio, e eu tenho que dizer o que eu vi. Esta deve ser a décima segunda vez que visitei um grupo de software bem sucedida, e existe uma grande semelhança entre os que realmente produtivas. Eu sei que muitas organizações querem trabalhar de uma forma extremamente rápida, produtiva, por isso estou indo para capturar um presente para os outros que querem fazê-lo, também. O pessoal da

CC foram articulada sobre o que eles fizeram, então eu acho que posso responder suas perguntas.

O projeto consiste em três pessoas, Kim, Pat e Chris. Kim é o líder da equipas e designer-chefe.

Kim diz que eles executar todos os projetos com duas a quatro pessoas, se possível, porque isso é quantos eles podem coordenar facilmente. Eles correm seis pessoas de vez em quando, mas isso é um exagero e eles se recusam a ir maior do que isso. Eles dizem que se eles precisam de mais de seis pessoas, eles não criaram seu escopo do projeto corretamente.

Todos eles se sentar em um quarto, como na foto. Eles fazem isso eles sempre que possível, mas se, por algum motivo, eles têm

usar escritórios separados, eles arranjá-los para ser ao lado do outro.

contato muito próximo é uma das coisas mais importantes para a sua forma de trabalhar. Eles querem comunicar para ser tão fácil como chamar outro lado da sala. Há muita coisa acontecendo para eles ter que pegar o telefone ou a pé no corredor. Eles podem fugir com escritórios adjacentes, mas apenas mal.

Eu perguntei o que eles usam para as suas necessidades e médio design. Eles apontaram para o quadro, e disse: "Não é o nosso meio de design."

Usando o quadro como o meio principal do projeto é uma das coisas que eu encontrei mais e mais. Um gerente de projeto me disse, e foi em um projeto muito maior, que ele estava tão pressionado pelo tempo que eles só fotografou o quadro branco como sua documentação design! Outra vez o líder da equipas mostrou-me as tabelas, linhas e rabiscos em seu quadro branco e disse: "Eu te desafio a me dizer o que publicou metodologia suporta esta notação!"

Eu tenho que concordar. Tenho visto muitos esboços em quadros em todo o mundo. Eu mal posso pensar em um meio melhor. A única coisa é que você não pode dobrá-la e levá-la com você! Graças a Deus existem imprimir quadros e quadros ligados ao PC, o dispositivo de rádio Mimeo para quadros comuns, e digitais máquinas fotográficas 5. Eu poderia facilmente custo justificar qualquer daqueles em tempo de comunicação salvo.

Kim disseram que usam casos de uso muito leve como uma base para os requisitos, vou enviar-lhe um exemplo de seus casos de uso Parágrafo Segundo mais tarde.

Na ocasião, eles acham que podem viver com requisitos ainda mais breves. Eles têm muito bons links para seus utilizadores, então as histórias mais leves o são na sua maioria apenas um lembrete de que devem ser desenvolvidas. As conversas contínuas com os utilizadores a manter as exigências precisas.

Uma vez que eles recolher os seus casos de uso e outros requisitos, eles não criam muitas outras prestações arquivados exceto o código comentado final e o texto da ajuda do utilizador. Eu sei que isso pode parecer incrível, mas eu já vi isso uma e outra e outra vez, nos grupos de maior produtividade que eu encontrei, e eu não acho que isso pode ser ignorado.

Eles sustentam revisões de projeto no quadro branco. Não há escrita formal para estes, a maior parte da informação é transmitida nos movimentos de falar e de mão, com alguns diagramas de instância e diagramas de interação tiradas ao longo do caminho, e alguns outros diagramas 6. Algumas delas não corresponder a qualquer padrão de desenho publicado -, mas por que eles deveriam, se todos na sala entende o que eles significam?

O que mais há a dizer?

---

5 O produto de software *Pixid* remove olhares de fotos de lousas digitais.

6 Veja o capítulo Produtos de Trabalho.

Eles olham uns sobre os outros ombros muito, às vezes fazendo *lado-a-lado de programação 7,* assim que receber um monte de código revendo feito em uma base contínua. Eles gostam disso, chamando-o "código de perscrutar." Eu tenho que correr em alguma forma ou de outra este em muitos lugares.

Extreme Programming (XP) as pessoas tomam a ideia mais longe, a programação em pares. XP'ers dizem que pegar mais erros dessa maneira, mais cedo, do que poderia imaginar, e os desenhos resultantes são melhores. Também lhes dá algum apoio intelectual. Há sempre alguém que entenda um pouco de como o código de cada pessoa é construído, e eles também aprender truques do outro de programação.

Eu pedi Kim e companhia se eles usaram quarto ou DOOM ou UML ou OML ou qualquer uma das outras coisas metodologia publicados. Eles não foram de todo tímido em um presente! Pat disse: "Nosso trabalho é fazer software para fora, não gastar o nosso tempo tirando fotos!" Chris disse: "Em quem devemos desenhá-los e por quê? Os nossos utilizadores ver os resultados tão rapidamente, e todos nós entendemos o interior completamente. Você já viu nossos quadros." Kim acrescentou: "Nós tentamos-los por um tempo, mas estamos tão estreitamente ligados que não acrescentam nada."

Este era um grupo muito franco, mas eu encontrei esta reação de outras equipas, e não apenas pequenas uns. A maioria das pequenas equipas são um pouco de desculpas sobre não usar as notações e ferramentas de desenho, como se eles estão fazendo algo errado. Por outro lado, um arquiteto em um projeto de 150 pessoas disse que duvida que as técnicas gráficas escala, e assim por diante o seu projeto grande que manteve todas as suas informações de design no texto.

Bem, aí está. Sem dúvida, você tem alguma dúvida, então fogo imediato e eu vou ver o que posso responder.

alegremente, Crystal

Caro Cristal,

Obrigado pela descrição. Eu acho que eu tenho uma pergunta para começar.

Conseguiu detalhes suficientes para descrever seu processo? O que

aconteceu e o que eles fizeram, e o que aconteceu em seguida, e o que eles fizeram?

melhores cumprimentos, Alistair

2 de junho: **Caro Alistair,**

Sim e não. I preso em torno de vários dias, para que eu pudesse vê-los em ação. O que eu diria que é isso, e você pode fazer com ele o que quiser. . .

Seu processo de minuto-a-minuto era tão complicado que eu não poderia escrever tudo, e se eu pudesse, você não poderia segui-lo, e muito menos instalá-lo em outro time.

O que eu diria é que eles se esforçam para desenvolver uma mente comunidade. Sentaram-se no início e compreendeu o problema que eles estavam resolvendo, funcionou como eles estavam indo para ir sobre ele,

7 Colocando suas telas de cerca de 18 polegadas separados, cada um eles trabalham em suas próprias tarefas de programação.

e, em seguida, começou a trabalhar. Eles estão em comunicação perto o suficiente de que eles sabem o que é importante eo que está acontecendo. Isso é suficiente para mantê-los em sintonia com o outro, e eles permitem que a dança, por assim dizer, improvisar-se em torno deles.

Aqui está o que eu posso oferecer:

Primeiro, eles entrevistam seus utilizadores e seu executivo patrocinadora. Pat também fui e vi os utilizadores no trabalho, então Pat se tornou o especialista local em ações do utilizador. Eles coletaram requisitos como casos de uso de dois parágrafos, você se lembra. Eles priorizado estes com os utilizadores, para que eles sabiam quais entregou o maior valor e foram necessários que aqueles em primeiro lugar.

Então eles tem seus requisitos de tecnologia em linha reta, o que eles tinham de usar e que eles poderiam usar. Kim é o líder técnico, então Kim praticamente decidido sobre a tecnologia e estabeleceu a arquitetura básica.

Terceiro, eles fizeram um plano de entrega, que incluiu uma entrega a cada dois meses.

Dentro de cada entrega, eles primeiro discutir o propósito de telas e seqüências de telas com utilizadores, usando apenas palavras ou lápis e papel, até que eles são bastante claras sobre as necessidades dos utilizadores. Eles discutem como configurar os objetos de negócios centrais e quem recebe o quê partes do sistema.

Normalmente, Kim leva a infra-estrutura, Pat leva a interface do utilizador, e Chris leva o negócio objetos. Eles não estão penduradas sobre isso. Houve momentos em que Pat e Chris cada um tomou ambos os objetos de negócios e objetos de interface do utilizador, e eles compartilharam frameworks.

Há uma demonstração inicial para o utilizador, depois de alguns dias ou algumas semanas, usando apenas a tela e alguns dummied de dados. Uma demonstração totalmente funcional vem depois talvez um mês. Todo mundo aprende alguma coisa com estes, apesar de terem estado em discussão muito de perto.

Eu vou te dizer o que aconteceu uma vez, só para você entender.

Chris tinha conseguido todas as telas de utilizador bastante bem certo e tudo, mas o primeiro protótipo levou sete segundos de ida e volta para uma transação. O utilizador olhou para a tela e perguntou: "Uh, é que a forma como ele vai realizar na vida real?" Chris disse: "Bem, nós temos algum ajuste de desempenho para fazer, mas mais ou menos, sim. Por quê?" O utilizador respondeu que ela é enviado um fax com centenas de linhas sobre ele, e ela simplesmente tipos do fax, 'cabeça para baixo', por assim dizer, nunca olhando para a tela, mas apenas tocar digitando as centenas de entradas. Ela não poderia esperar sete segundos entre cada linha.

Chris ficou pasmo. "Mas o que você faz se há um erro de digitação?" "Oh, bem, este é todos registrados apenas para fins de dedução fiscal, por isso, na verdade, nós não muito cuidado se houver um erro. Não vale a pena o tempo para corrigi-lo", respondeu o utilizador.

Chris foi bastante abalado por quanto eles tinha entendido mal os requisitos, mesmo depois de entrevistar o utilizador e mostrando esboços das telas e tudo. Eu acho que isso só serve para mostrar a diferença entre uma maquete e uso real.

Essa é a maior parte do "processo".

aplausos, Crystal.

Caro Cristal,

Obrigado pelas notas de campo. Estou ansioso para sua próxima parcela. Diga-me sobre o teste. Será que eles têm ferramentas de teste, testes de regressão, testadores externos, o que eles fazem?

graças, Alistair

03 de junho: Caro Alistair,

Sim, eu queria chegar a testar cedo ou mais tarde. Primeiro deixe-me terminar o processo, e o teste vai aparecer.

Imagine que eles têm uma período de entrega com duração de 2 meses. Eles vão nos requisitos com os utilizadores, e começar a elaborar o seu design. Aqui é aonde o processo fica confuso.

Algumas das pessoas em CC são "pensadores", outros são "typers."

Os pensadores sentar e pensar sobre o problema, projetar no papel ou quadro branco durante o tempo que eles podem estar, até que quer ter um design em que acreditam, ou então eles têm que começar a programar para trabalhar a sua confusão. Esse pensamento, por vezes, leva alguns minutos, outras vezes leva horas, ou até mesmo um par de dias no pior caso. Depois que eles programar algum código real, eles aprendem mais sobre como seu projeto está funcionando, e que quer continuar com ele, ou pensar e desenhar um pouco mais sobre como ele precisa ser mudado.

Os "typers" basta começar a digitar imediatamente. Pelo menos, é assim que parece. O que eles realmente fazer é pensar sobre um pequeno pedaço de funcionalidade, escrever um teste de unidade se eles estão fazendo o desenvolvimento orientado a testes e, em seguida, obter essa pequena seção do código de trabalho. Quando adicionar a próxima seção, que refatorar os dois juntos para melhorar o design. Essas pessoas passam mais tempo digitando e menos de encarar que os outros, mas eu não posso particularmente dizer que não faz qualquer diferença real no longo prazo. Parece ser mais uma coisa personalidade.

testes constantes. O melhores grupos, e CC é um deles, deliberadamente projetar uma arquitetura que suporta testes de regressão totalmente automatizado. Alguns grupos de executar seus testes a partir da janela do espaço de trabalho, em seguida, colocá-los direto para o sistema de classes com o resto do código de produção. Outros grupos de colocá-los em arquivos e executar trabalhos em lote contra o seu código, puxando os casos de teste para fora dos arquivos.

Kim disse que é difícil convencer as pessoas, mesmo experientes incomodar a criação de uma arquitetura de sistema que suporta testes de regressão, para construir casos de teste como eles vão. No entanto, uma vez que experimentar o prazer de desenvolver com testes de regressão automatizados na ponta dos dedos, eles nunca vão

de volta. A arquitetura testes de regressão e lote de teste dá-lhes uma liberdade de fazer alterações e uma sensação de segurança que eles nunca tiveram antes. Eles fazer algumas mudanças, em seguida, empurre o botão de teste e descobrir se eles se atrapalharam com algo que estava trabalhando antes.

Vamos ver, processo. . . Eles têm uma série de pequenas revisões de projeto dentro da equipas, eles demonstração do sistema para os utilizadores um par de vezes, construir testes de regressão, e em algum momento eles declaram que eles são feitos. Eles escrevem o manual do utilizador ou o texto de ajuda, criar uma compilação de produção, execute os testes novamente e, em seguida, entregar o sistema para alguns utilizadores amigáveis

Como eu estou indo?

Cristal

Caro Cristal,

Indo bem. Agora, cada metodologia deve anunciar suas tolerâncias. Você está dizendo que Crystal Clear tem tolerâncias como este:

- Codificação estilo tem grande tolerância, sendo um mater para a equipas para decidir.
- Por outro lado, testes de regressão, peering código de pares, links do utilizador, lançamentos curtos são questões com muito pouca tolerância.
- tamanho da equipas tem alguma tolerância, a gama sendo 2-8 pessoas.
- comprimento versão tem alguma tolerância, a faixa a ser 1-3 meses. Estou com você sobre essas coisas. Eu ainda estou preso no processo, no entanto. isto

Parece que eles apenas "correr e fazer *coisa.* " Eu meio que começa a idéia, mas eu não consigo pensar em como descrevê-lo para outra pessoa.

O que você acha que devemos fazer em relação a esta falta de descrição?

Confiando em você, mas ainda confuso, Alistair

4 de junho: Caro Alistair,

Levar a sério pessoas, Alistair, estes são cultivados, e ter equipamento mental normal. O que eles fazem? Eles olham ao redor, eles pensam, falam uns com os outros, é o que eles fazem.

Uma nota sobre o líder da equipas ou designer-chefe, no entanto:

Kim disse que eles sempre têm uma pessoa experiente na equipas. Eles têm muitas vezes um novato. Se eles têm dois noviços de classificação, o líder da equipas deve ser particularmente experiente e bom.

Lembro-me que você me contou sobre seu primeiro trabalho recém-saído da faculdade, e não era muito diferente. Você estava em uma equipas de três projetar um subsistema de um simulador de vôo hardware. Você era o novato. Partilhou o escritório com o seu chefe, o líder da equipas e designer-chefe. O outro desenhador sentou-se no escritório através de você. Seu chefe projetado arquitetura do subsistema e outros dois contribuíram projeto e ideias.

Seu líder de equipas tinha passado por isso antes, e seu chefe tinha sido através de mais. Você viveu e aprendeu. E isso foi em um projeto com 26 estilistas total. É muito mais fácil se o

todo o projeto é de apenas três a cinco pessoas.

Você está disposto a confiar em que três pessoas inteligentes, treinados podem usar seu senso comum?

A principal coisa é que o processo global oferece lançamentos muitas vezes que há pouco tempo para o projeto ir muito longe fora da pista.

Então lá.

Você fala sobre técnicas. Bem, essas pessoas utilizam todas as técnicas conhecidas para a ciência da computação, e um monte de coisas não descrita. É por isso que esta metodologia tem que ter um monte de tolerância. Eles se preocupam com a forma como o resultado final parece, não o que seqüência alguém passou por inventá-lo.

aplausos como sempre, Crystal

Caro Cristal,

Uh, obrigado por isso, eu acho que eu precisava. De alguma forma ele é tão fácil de esquecer-se sobre as pessoas pensando e falando uns com os outros ao fazer essas coisas processo. :-(

Na verdade, I * am * querendo saber como fazer isso ISO / CMM (I) certificável 8, mas vamos deixar isso por um tempo e descobrir o que mais há a dizer.

OK, você falou sobre a configuração da equipas, conexões de utilizadores, desenvolvimento iterativo incremental, casos de uso, resultados, os testes, o que mais está lá para cobrir?

regards, Alistair

5 de Junho: Caro Alistair,

Você esqueceu de mencionar algo extremamente importante em uma metodologia: os valores da equipas. Que impulsiona o resto da metodologia.

A frase chave aqui é "luz sobre produtos de trabalho, forte na comunicação", com *melhoria reflexiva.* Eles trabalham no desenvolvimento de uma mente comunidade, com ligação constante para os utilizadores e manifestações e entregas frequentes. O marco favorecido é entregue funcionalidade, e com isso eles podem corrigir qualquer problema.

Ao ficar luz sobre os produtos de trabalho, eles se movem muito mais rápido, e mudar de direção mais rápido. Uma vez que eles podem se mover mais rápido, eles podem oferecer algo útil em menos tempo, tornando assim as suas metas. Uma vez que eles entregar em menos tempo, eles precisam de menos produtos de trabalho intermediários. Ele é auto-reforço, como você vê.

Eles vêem o desenvolvimento de software como uma conversa em evolução - um jogo económico-cooperativo, na verdade. Em qualquer movimento que vá em frente e criar lembretes para o outro.

Isso é suficiente para satisfazê-lo?

alegremente cristal

Caro Cristal,

Isso é ótimo, obrigado. Vou ver se eu posso capturar o que você disse em algum tipo de formato fechado para que alguém possa ir embora com ele no papel.

8 Veja a análise do auditor em *Testado.*

Devo usar a lista de elementos de metodologia que utilizo em meus outros livros, com papéis, as entregas, equipas, habilidades, normas, e tudo isso? Receio que vou ter problemas com toda essa tolerância que você quer que eu ficar lá.

O que realmente me impressiona sobre isso é que quando você escrever para baixo como uma organização funciona, mesmo um pequeno grupo com uma metodologia leve e pequeno conjunto de resultados, ele realmente parece complicado!

melhores cumprimentos, Alistair

06 de junho: **Caro Alistair,**

Eu vou propor algo mais curto, mais estável e mais fácil de lembrar do que suas tabelas e tuplas. Eu acho que alguns dos itens em suas tuplas não são realmente crítico, e há vários caminhos que trabalham.

Como cerca de incidindo sobre a chave *Propriedades* puseram no lugar, e deixe equipas diferentes para seguir seus próprios caminhos preferidos, apenas contanto que eles estabelecer as propriedades.

Por exemplo: Entrega Frequent como uma propriedade. Como é frequente Frequente, e exatamente o que conta como um Entrega? Como eles fazem a entrega? Proponho que deixar as respostas até a equipas para decidir, desde que Entregas frequentes de algum tipo estão acontecendo. Alguns dos outros seria:

Melhoria reflexivo osmótica

Comunicação Segurança

Pessoal

Foco

Fácil acesso a utilizadores experientes e não se esqueça sua

Ambiente técnico, com testes automatizados, gestão de configuração, integração frequente.

Para os produtos de trabalho, fornecer *amostras.* Eles vão ser muito mais útil do que modelos. Obter fotos de quadro branco, esboços, qualquer que seja, de equipas reais.

Sim, até mesmo uma pequena metodologia é uma coisa complexa, mas o que você espera? É a codificação de uma cultura e de uma cultura de sucesso que produz software sério. Crystal Clear é tão simples e curto como eu pode mantê-lo, mas não posso fazê-lo mais simples do que é.

Apenas deixe ir as técnicas.

alegremente cristal

Caro Cristal,

Entendi. Vamos considerar técnicas de uma "questão local", então.

Aqui está a pergunta seguinte. Muitas pessoas vão pensar que Crystal Clear só irá funcionar com todos os desenvolvedores experientes.

O que você tem a dizer sobre isso?

Atenciosamente, Alistair

07 de junho: **Caro Alistair,**

questão relevante. Resposta longa . . . Pessoas aprendendo uma nova habilidade passar por três bastante diferentes estágios de comportamento, conhecido como o *seguinte, destacando,* e *fluência.* Podemos chamá-los de níveis 1, 2 e 3. Esses estágios de

aprendizagem são bem conhecidos no aikido, aonde são chamados lá *Shu, Ha,* e *Ri,* o que se traduz, grosso modo, *siga, quebrar, partir.* Provavelmente não é um acidente que casas de artesanato na Idade Média também tinha três níveis: aprendiz, trabalhador qualificado e mestrado.

Isso tudo é descrito no *Desenvolvimento Ágil de Software* mas é talvez uma boa idéia para resumir aqui, também.

No Japão, eles desenham Shu-Ha-Ri com os personagens:

Figura 1-2. Shu, Ha, Ri. Uma pessoa apenas começando não pode aprender vários métodos de uma vez, mesmo se houver dez que poderia funcionar. Então, no Nível-1, o *Segue* estágio, as pessoas procuram *1*

procedimento que funciona. Eles copiá-lo. Eles aprender. Eles medem o sucesso pela maneira como eles podem segui-lo. Na verdade, eles tomam como certo que ele funciona, e pode ficar muito chateado se não sobre eles.

o *Shu* estágio manifesta-se em metodologias de desenvolvimento de software como manuais grossos e detalhadas.

Este livro é um exemplo. Crystal Clear pode ser descrito em uma frase (como no *Destilada!).* Para um desenvolvedor de software experiente, esta frase é uma orientação adequada.

Essa frase não é suficiente para um praticante de Nível-1. Ele quer detalhes sobre o que fazer. Quando eu expandir um especial técnica, processo ou produto de trabalho, estou aos leitores no Nível-1 sobre esse item particular.

Todo mundo é Nível-1 em uma nova habilidade. Mesmo desenvolvedores avançados trabalhar no Nível 1 da primeira vez que tentar um Workshop de Reflexão.

Eventualmente, eles reconhecem que a técnica não vai servir todas as situações.

No *destacando,* ou Nível-2 estágio, eles procuram regras sobre quando o procedimento quebra. Deixe-me dar dois exemplos.

Na década de 1990, as organizações que se tornaram altamente sintonizado para Engenharia da Informação (IE) arquiteturas de repente tinha que entregar software orientado a objetos. Eles primeiro tentou adaptar os métodos de IE, em seguida, eles desenvolveram completamente novas metodologias de desenvolvimento. Ao longo do caminho, muitas vezes eles regrediram através totalmente *Ad hoc*

técnicas antes de descobrir novas estruturas para apoiar os projetos OO.

A segunda mudança aconteceu depois do Manifesto de Desenvolvimento Agile foi publicado em 2001 9. Muitos grupos que tentam para fora encontrada uma incompatibilidade entre ágil e suas práticas organizacionais padrão. Então, eles experimentou com ideias alternativas.

Eventualmente, as pessoas tornar-se fluente em situações suficientes que parar de prestar atenção para que a técnica que eles estão usando a qualquer momento. Eles inventam,

9http://AgileManifesto.org

misturar, adaptar tudo o que sabem mais rápido do que eles podem até mesmo descrever.

Isto é o *fluência,* ou fase Nível-3. desenvolvedores de nível 3 não pagam grande atenção à fórmula metodologia. Eles prestam atenção ao que está acontecendo no projeto e fazer ajustes de processos em tempo real. Eles entendem o efeito final desejado e simplesmente fazer o seu caminho para esse fim. Basta indicar o desejado *propriedades* do projeto é suficiente para eles.

As discussões entre os de nível 3 pessoas soar distressingly Zen:

"Use uma técnica, desde que ele está fazendo algo de bom."

"Faça o que funciona."

"Quando você está realmente fazendo isso, você não tem conhecimento de que você está fazendo isso."

Para alguém no nível de fluência, tudo isso é verdade. Para alguém ainda destacar, é confuso. Para quem procura um procedimento a seguir, é inútil.

A descrição de uma frase de Crystal Clear é o suficiente para o desenvolvedor Nível-3. Essa descrição foi usada diretamente por vários líderes de projetos, um dos quais relatados de volta mais tarde, "Fizemos o que você disse, e funcionou!" (Que, para um público Nível-1, não se comunica muito).

Tente estar ciente destes níveis na próxima vez que você está em uma reunião. Você é muito provável para encontrar duas pessoas presas em um argumento, e notar que eles estão em níveis diferentes sobre o tema atual. O que é um conjunto natural de variações para uma pessoa parece bewilderingly vaga para o outro. Se você identificar a diferença de nível, os dois podem negociar o nível da discussão, ou pelo menos ser mais paciente com os outros.

Tudo isso se aplica a Crystal Clear. Eu não acho que Crystal Clear pode ser executado por uma equipas de todos os praticantes de Nível-1. Duvido que essas 300 páginas de descrição são suficientemente explícito.

Tom Poppendieck descreve o paradoxo ordenadamente: Nível 1 as pessoas precisam de detalhes, mas não pode lidar com sua complexidade. Nível 3 pessoas podem lidar com a complexidade, mas têm pouca utilidade para o detalhe.

Para lidar com esse paradoxo, tenho em Crystal Clear que qualquer um projeto precisa de pelo menos um desenvolvedor Nível-3 ou dois desenvolvedores Nível-2. Naquela *Designer-chefe* expande a informação em tempo real. Estas páginas apoiar ele ou ela em fazer isso.

Na prática, o designer-chefe mais ou menos executa o projeto. Muitas vezes, há uma outra pessoa bastante experiente, mas o resto são todos em vários estágios de início ou mediano (alguns jovens, inexperientes e brilhante, alguns mais velhos e cansados ou apenas em seu nível máximo de capacidade).

Então, finalmente, eu posso responder à sua pergunta: Não, não é o caso que Crystal Clear só irá funcionar com todos os desenvolvedores experientes. Mas pelo menos um deles precisa ser um desenvolvedor de nível 3.

Tenho em mente que o patrocinador do projeto será a contratação de acordo com esse padrão de pessoal.

Qual é a sua próxima pergunta? Cristal

Caro Cristal,

Obrigado, e sim, eu posso já dizer aonde eu posso usar esses níveis de desarmar algumas reuniões.

Parece-me, de repente, que você não disse muito sobre ferramentas até agora. Sinto que há mais aqui do que você deixou, e que é realmente muito importante.

regards, Alistair

08 de junho: **Caro Alistair,**

Sim, ferramentaria merece menção especial.

Limpar não *mandato* quaisquer ferramentas específicas, da mesma forma que não impõe qualquer técnica específica. Isso é porque as ferramentas são diferentes em várias tecnologias.

Dito isto, há uma ferramenta que são mencionados por quase todos os time que eu visitar, incluindo até mesmo as menores equipas de três pessoas: a ferramenta de gestão de configuração.

Um número surpreendente de equipas que visitam não têm qualquer sistema de gestão de configuração no lugar. Essas não são as equipas de sucesso estou documentando, no entanto. A ausência de gestão de configuração não implica o projeto irá falhar, mas equipas de sucesso, quase por unanimidade nomear sua ferramenta de gestão de configuração como a ferramenta mais importante que possuem.

Cristal

Caro Cristal,

Última pergunta difícil --- O que significaria para um grupo para dizer que estão "em conformidade" com esta metodologia?

Alistair

09 de junho: **Caro Alistair,**

Compliance é uma coisa engraçada. Por um lado, queremos o cumprimento suficiente para que o projeto é seguro, e não tanto que as pessoas ficam tensos sobre os detalhes; orientação para equipas interessadas na criação de seus projetos para ter sucesso, trabalhar de forma eficiente, e ser "habitável" ao mesmo tempo.

A conformidade com metodologias é um declive gradual. Imagine, se quiser, um *mesa,* um planalto acima algumas planícies. Mas imagine que em vez de um penhasco das planícies à mesa, há uma inclinação inclinado.

A mesa no topo é "obviamente compatível." Há uma certa quantidade de espaço para o movimento, tudo com segurança dentro a letra e a intenção da metodologia.

A inclinação é a zona de transição, a área cinza de conformidade, "Estamos compatível? Não é a letra da metodologia, mas é na intenção?"

Uma descrição metodologia deve dizer algo sobre a zona de transição, caso contrário, haverá incessante

discutindo sobre a carta contra a intenção da metodologia. Ele deve dizer algo sobre: "Aqui está o que

é preciso dizer que você está seguro, aqui estão algumas coisas que você pode fazer e ainda digo que você está fazendo (ou tipo de fazer) ele; fazer estes e você está definitivamente fora."

A linha divisória na parte inclinada nunca será perfeitamente claro, mas deve haver alguma maneira de dizer o deslocamento entre as três zonas. Isso permite que as pessoas dizem, "Gee, estamos fazendo tudo exatamente como descrito, e ele ainda não está funcionando", ou "Oh, eu vejo. Esquecemos *este* parte crítica. Vamos tentar isso e ver se tudo funciona melhor."

Assim, aqui é o que é preciso dizer que você está usando Crystal Clear. Porque estes são grandes, eu começo cada um com um rosto sorridente :-)

1. **: -) Você tem suma, rico caminhos de comunicação. Você está sentado muito perto juntos, de preferência uma sala grande. Você pode estar sentado em salas adjacentes, se há realmente apenas 4-6 de você e você visitar um ao outro muito. Reconhecer que a qualidade da comunicação cai do momento que você tem que pegar o telefone ou a pé para fora da porta.**

2. **:-) Você entregar incrementos mensal ou a cada dois meses, absolutamente não mais de 3 meses de intervalo. Você agendar e acompanhar marcos execução de código, e não por marcos de conclusão do documento.**

3. **:-) Você tem um utilizador real no projeto, mesmo que apenas a tempo parcial. Seu utilizador ajuda você a construir seus esboços de tela e ambos valida e quebra seus projetos de interface do utilizador. Você começa o sistema revisado por utilizadores reais, pelo menos uma vez por incremento antes de entregá-la.**

4. **:-) Você tem uma declaração de missão do projeto. Você usar um de uma ampla variedade de formatos de requisitos. Você tem uma descrição do projeto do sistema (usando uma das muitas técnicas de descrição você pode inventar).**

5. **:-) Você tem um modelo de propriedade claro para seus produtos de trabalho. Para cada classe, módulo ou caso de uso, você sabe quem é que pode mudar ou, mais importante ainda, excluir partes dela. A resposta pode ser tanto uma pessoa única, qualquer um em uma determinada sub-time, ou mesmo alguém na equipas (esta é a regra em projetos XP). Você nunca deve ter para ter a equipas inteira se sentar ao redor da tela e perguntar: "Quem colocou isso aqui? O que ele está fazendo lá? Será que podemos excluí-lo?" Se você tiver que fazer isso, você sabe que tem uma ruptura no modelo de propriedade.**

Aqui estão algumas coisas que você pode variar e ainda dizer que você está fazendo claro. Porque estes são ainda OK, eu começo cada um com uma face neutra: - |

1. **: - | Você senta-se em salas adjacentes, se há realmente apenas 4-6 de você e você visitar um ao outro muito. Reconhecer que a qualidade da comunicação largar o momento em que você tem que pegar o telefone ou a pé para fora da porta.**

2. **: - | Você usa a técnica Scrum backlog ou jogo de planificação do XP para priorizar e agendar o seu trabalho (Schwaber 2001 (Beck, 2000). Eu pessoalmente gosto de usar o *bl i tzplanning (* também conhecida como a sessão técnica projeto- planificação jam). O ponto que desejo fazer aqui é que Crystal Clear permite empréstimos de outras metodologias. Usando XP de *jogo do planificação* não significa que você não está fazendo Crystal Clear. Significa apenas que a incorporação de elementos de outras metodologias é uma parte natural do Crystal Clear.**

3. **: - | Há apenas uma iteração por entrega, mas você organizar várias sessões de utilizador. Quando os utilizadores vêem o sistema, eles vão comentar, o que irá gerar um novo trabalho a ser colocado no ciclo de entrega. Que o feedback é crucial para o sucesso do produto. "Iterações" fornecer encerramento e informação de velocidade, mas assim como os ciclos curtos de entrega. Nada substitui as visões de utilizadores, no entanto.**

4. **: - | Você os utilizadores não podem aceitar incrementos operacionais tão frequentemente como você produzi-los, ou eles não podem aceitar funcionalidade parcial. Assim, Você entrega testado código, entrega-pronto para uma "estação de preparo", aonde o incrementos são adicionados em conjunto ao longo do tempo até que seja adequado para libertá-los para os utilizadores finais.**

5. **: - | Você descreve o seu projeto usando documentos de hipertexto e memorandos técnicos, e evitar desenhos UML .; ou, você descreve classes usando UML ou uma das notações -OOM, e evitar demonstrações de responsabilidade. Você documentar seu sistema usando vídeos dos designers andam através do design. Há muitas variações de como descrever a concepção (não estamos nem perto de descobrir a melhor maneira). Você não gosta do meu formato de caso de uso, e chegar a sua própria formato de requisitos (possivelmente histórias de utilizadores ou listas de recursos). Com a maioria dos produtos de trabalho, a substituição equivalente é permitida, desde que a intenção do item é preservada e propriedades de segurança do projeto não estão em perigo.**

Aqui estão algumas coisas que, se você está fazendo, você está definitivamente fora da Clear. Seu conjunto de regras pode funcionar para você, mas não é Crystal Clear. Eu começo cada um com uma face frowny :-(

1. **:-(Você trabalha em edifícios diferentes ou cidades diferentes. Isso significa que seus caminhos de comunicação são longos e finos, que você deve se comunicar por telefone, e-mail ou visitas ocasionais. Você não pode cumprir os requisitos de Crystal Clear, o que exige**

este membros da equipas ser capaz de iniciar e realizar conversas de pessoa a pessoa muitas vezes ao dia com pouco esforço. distâncias mais longas levantar a barreira para a conversa, a forma de comunicação é inferior, e você não pode confiar na memória compartilhada comum do grupo. (Noto, como fará provavelmente, que trabalhando em diferentes andares do mesmo prédio cai em uma zona cinzenta nestas descrições. Eu duvido que você pode fazer Limpar trabalho corretamente em andares. Eu acho que você não pode fazê-lo funcionar corretamente, mesmo para baixo um corredor. Mas deixo-me aberto à refutação pelo exemplo.)

2. **:-(seus incrementos são mais de 4 meses. Seu feedback caminho é muito longo. Você não pode mais evitar "notas promissórias" tipos de documentos, porque você não está entregando muitas vezes suficiente para demonstrar o progresso que você está fazendo. incrementos mais curtos é um fator crítico de sucesso para o seu projeto (ver *Sobreviver orientada a objectos rojetos P),* não importa o que metodologia você está usando.**

3. **:-(Não há utilizador visualiza antes da entrega. Minha experiência é que isso significa que os utilizadores será dado algo que não atende às suas necessidades, demasiado tarde para fazer alterações. Mesmo projetos que fazem tudo o resto na lista Crystal Clear, que perca isso, acabar em problemas quando os utilizadores finalmente fazer ver o sistema.**

4. **:-(Não há utilizadores reais disponíveis para a sua equipa. O risco a ser abordado aqui é que as necessidades dos utilizadores são provenientes de algum tipo de intermediário, uma representante de marketing ou executivo que promete eles estão dizendo o que os utilizadores "realmente" quer. A pesquisa indica que este tipo de entrada do utilizador não é confiável, e são necessários links diretos para os utilizadores reais.**

Há uma parte da Clear que está entre opcional e obrigatório: o uso de um totalmente automatizados suites equipamento de teste de regressão e de teste para testes unitários e de aceitação. Ela recebe o? símbolo.

1.? Não há nenhum teste automatizado suites. Muitos sistemas têm enviado com sucesso sem conjuntos de testes automáticos. as pessoas realmente bem sucedidos, por outro lado, repetidamente dizer que os conjuntos de teste dar-lhes velocidade e conforto em melhorar e depurar o sistema, e está no topo de sua lista de prioridades. Estes dias, a velocidade em código alteração é dependente de ter automatizado conjuntos de testes disponíveis, e mais equipas estão usando test-driven development (Beck 2002) para fazer ainda melhor.

As equipas que usam testes automatizados acham que podem atualizar o design do código com grande liberdade: Eles testar novamente o seu trabalho com o premir de um botão, descobrir se eles introduziram um novo bug ou não.

Evitar a introdução de novos erros ao remover os antigos é uma das melhores maneiras de melhorar os prazos de entrega do projeto encurtar.

Sem testes de regressão, de fazer alterações no sistema torna-se tediosa e pouco confiável, não confiável o suficiente para que uma de duas coisas ruins acontecem: a equipas faz a mudança e introduz novos erros, ou não fazer a mudança, e permitir que o código a crescer anormalmente complexo ("intrincada" é uma palavra usada frequentemente para tal código).

Por que é automatizado testes de regressão opcional no Crystal Clear se é tão grande?

A resposta é que os testes de regressão automatizados não são realmente necessários para ter um sucesso do projeto. Ele torna a vida da equipas mais fácil, torna o código mais maleável, mas eu vi e participou em muitos projetos que sucederam sem usar suites totalmente automatizado de teste de regressão.

Desde Crystal Clear é proposto como uma metodologia de alta tolerância, eu tenho que sair testes de regressão automatizado como opcional. Em um nível pessoal, posso dizer que os melhores programadores que conheci usá-los e recomendar altamente. . . mas não usá-los não significa que o projeto irá falhar, e por isso não é uma base para dizer um grupo não está usando Crystal Clear.

Então, você tem isso. O que é preciso para ser com segurança dentro claro, o que pode variar, e que é preciso para estar fora.

Caramba, eu estou exausto! Quanto mais você quer de mim?

Cristal

Caro Cristal,

Eu prometo esta é a última pergunta - Atualizar para mim o que Crystal Clear está tentando ser e por que funciona?

Alistair

10 de junho: **Caro Alistair,**

Resumindo. . . Crystal Clear pretende ser um conjunto simples e tolerante de regras que coloca o projeto na zona de segurança. É, por assim dizer, *a mínimo metodologia que poderia funcionar,* porque contém tanto tolerância.

Toda a família de cristal tem sido descrito como consistindo de

- **As entregas frequentes,**
- **Melhoria reflexivo, e**
- **Fechar Comunicação. Crystal Clear empurra esse último até osmótica Comunicação.**

Core para toda a família de cristal são um par de ideias. Uma delas é que os produtos de trabalho intermediários e "notas promissórias", tais como documentos de requisitos detalhados, documentos de design e planos de projeto não pode ser eliminado, mas eles podem ser reduzidos na medida em que

- , ricos, caminhos de comunicação informais curtas estão disponíveis para a equipas e
- de trabalho, software testado é entregue mais cedo e com frequência. Outra é que a quantidade de detalhes necessários nos requisitos, documentos de projeto e planificação varia com as circunstâncias do projeto, especificamente

- a extensão dos danos que pode ser causada por defeitos detetados e
- a frequência de colaboração pessoal.

Finalmente, a equipas reflete e ajusta suas convenções de trabalho em tempo real para caber

- as personalidades na equipas,
- o ambiente de trabalho local,
- a atribuição atual. A razão Crystal Clear ele funciona é que as pessoas envolvidas são profissionais liderados por alguém que é suposto ser um desenvolvedor de Nível-3. Com ricos, comunicações baratos entre eles, os membros da equipas irá diagnosticar a maioria de seus próprios problemas. Se eles têm acesso aos utilizadores e entregar a execução de código a cada mês ou dois, eles obter feedback sobre o que estão fazendo. Se eles não são interrompidos, muitas vezes, eles vão fazer progressos. Em outras palavras, o projeto pode vir a trabalhar.

Note-se que Crystal Clear não tem como objetivo ser a metodologia mais produtiva, rigoroso, ou escalável, além de seu objetivo de ser simples, tolerante e bem sucedida. As pessoas são bem-vindos a adotar hábitos de desenvolvimento mais disciplinados como quiserem, incluindo um utilizador em tempo integral, programação em pares, test-

desenvolvimento orientado, e testes de regressão totalmente automatizados.

Eles são, no entanto, pouco provável que seja capaz de soltar qualquer uma das regras de Crystal Clear e ainda ser seguro.

Crystal Clear vem com uma importante cláusula de verdade-em-publicidade: Oito pessoas ou menos, co-instalados, não um sistema crítico de vida. equipas distribuídas precisa de outras formas de comunicação, equipas maiores precisam de mais coordenação e equipas que trabalham em sistemas críticos da vida precisamos de mais procedimentos de verificação no local. Essas equipas terão de encontrar outra metodologia, ou sintonizar Crystal Clear para cima, para atender suas necessidades.

Eu acho que isso encerra nossa discussão. Os melhores cumprimentos, Crystal

Capítulo 2 Aplicada (As sete propriedades)

Leitura como Crystal Clear funciona levanta duas questões particulares:

> ***"O que são essas pessoas concentrando-se em enquanto eles trabalham?" "Podemos chegar mais longe na zona de segurança?"***

Este capítulo descreve sete propriedades criados pelas melhores equipas. Crystal Clear exige os três primeiros. Melhores equipas usam as outras quatro propriedades para obter mais para dentro da zona de segurança. Todas as propriedades para além de Comunicação osmótica se aplicam a projetos de todos os tamanhos.

Eu só recentemente despertou para a percepção de que consultores top trocar notas sobre a *Propriedades* de um projeto em vez de sobre os procedimentos seguidos. Eles inquirir após a saúde do projeto: "Existe uma declaração de missão e um plano de projeto que eles entregam frequentemente são o patrocinador e vários utilizadores experientes em estreito contato com a equipas??"

Por conseguinte, e em uma partida da maneira em que uma metodologia é geralmente descrita, peço equipas Crystal Clear a chave alvo *Propriedades* para o projeto. "Fazer Crystal Clear" torna-se atingir as propriedades em vez de seguir os procedimentos. Dois motivos conduzir esta mudança de procedimentos para propriedades:

- **Os procedimentos podem não produzir as propriedades. Dos dois, as propriedades são as mais importantes.**
- **Outros procedimentos que os que eu escolher pode produzir as propriedades para sua equipas particular.**

A família de cristal enfoca as três propriedades Entrega frequente, Fechar Comunicação, e Melhoria reflexivo 10 porque eles devem estar presentes em todos os projetos. Crystal Clear aproveita o tamanho da equipas pequena e proximidade para fortalecer

Fechar Comunicação para o mais poderoso Comunicação osmótica. Além de que um turno, desenvolvedores experientes vai notar que todas as propriedades que delineiam neste capítulo se aplicam a todos os projetos, não apenas projetos de pequena equipas.

Ao descrever Crystal Clear como um conjunto de propriedades, espero alcançar no *sentindo-me* do projeto. A maioria das descrições metodologia perca o sentimento crítico que separa uma equipa de sucesso de um fracassado. A equipas Crystal Clear mede sua condição pelo humor da equipas e os padrões de comunicação, tanto quanto pela taxa de entrega. Nomear as propriedades também fornece a equipas com frases de efeito para medir a sua situação: "Nós não fizemos qualquer Melhoria reflexivo por um tempo ..." "Podemos obter mais Fácil acesso a utilizadores experientes ". Os nomes de propriedade-se ajudar as pessoas a diagnosticar e discutir maneiras de corrigir a sua situação atual.

---

10 Graças a Jens Coldewey da Alemanha para apontar isto para mim!

Propriedade 1. **Entrega Frequent**

A única propriedade mais importante de qualquer projeto, grande ou pequeno, ágil ou não, é o de emitir execução, testado código para os utilizadores reais a cada poucos meses. As vantagens são tão numerosos que é espantoso que qualquer equipas não fazê-lo:

- **Os patrocinadores obter feedback crítico sobre a taxa de progresso da equipas.**
- **Os utilizadores tem a chance de descobrir se o seu pedido original para o que eles realmente precisam e para se suas descobertas alimentado de volta para o desenvolvimento.**
- **Desenvolvedores manter seu foco, rompendo bloqueios de indecisão.**
- **A equipas começa a depurar o seu desenvolvimento e processos de implantação, e recebe um impulso moral por meio de realizações. Todas essas vantagens vêm de uma única propriedade, Entrega frequente. Nas minhas entrevistas, eu não vi qualquer período superior a quatro meses que ainda oferece essa segurança. Dois meses é mais seguro. Equipas que implementam a web pode oferecer semanal.**

Você já entregues em execução, testado e código utilizável pelo menos duas vezes a sua comunidade de utilizadores em

nos últimos seis meses?

* * * *

Apenas o que significa "entrega" significa?

Às vezes, isso significa que o software é implantado para o conjunto completo de utilizadores no final de cada iteração. Isto pode ser prático com software de web-implantado ou quando o grupo de utilizadores é relativamente pequena.

Quando os utilizadores não pode aceitar atualizações de software que, muitas vezes, a equipas se encontra em um dilema. Se eles entregam o sistema com frequência, a comunidade de utilizadores vai ficar irritado com eles. Se eles não entregam com frequência, eles podem perder um problema real com a integração ou implantação. Eles vão encontrar esse problema quando é muito tarde - no momento da implantação do sistema.

A melhor estratégia que eu conheço nesta situação é encontrar um utilizador amigável que não se importa de experimentar o software, seja como uma cortesia ou por curiosidade. Implantar em que uma estação de trabalho. Isso permite que a equipas para a prática de implantação e obter feedback útil de pelo menos um utilizador.

Se você não consegue encontrar um utilizador simples para oferecer para, pelo menos, realizar uma plena integração e teste como se estivesse indo para. Isso deixa apenas de implantação com uma falha potencial.

* * *

Os termos *integração, iteração, visualização do utilizador,* e *liberação* se misturam juntos estes dias. Eles têm efeitos diferentes sobre desenvolvimento e deve ser considerado separadamente.

Integração Frequent deve ser a norma, a acontecer a cada hora, a cada dia, ou na pior das hipóteses, a cada semana. As melhores equipas estes dias têm funcionando continuamente scripts de build-e-teste automatizados, então nunca há mais de 30 minutos a partir de um check-in até o Teste automatizado afixação dos resultados.

Basta realizar uma integração do sistema não constitui uma *iteração,* desde uma integração muitas vezes é realizada após uma única pessoa ou subequipas completa como fragmento de um trabalho de programação. O termo *iteração* refere-se à equipa de completar uma seção de trabalho, a integração do sistema, relatando o resultado na cadeia de gestão, fazendo a sua periódica Melhoria reflexivo (Desejo), e muito importante, ficando fechamento emocional por ter concluído o trabalho. O fechamento após uma iteração é importante porque estabelece um ritmo emocional, algo que é importante para nós como seres humanos.

Em princípio, uma iteração pode ser em qualquer lugar de uma hora para um quarto. Na prática, eles são geralmente duas semanas a dois meses de duração.

A data final de uma iteração é geralmente considerado imóvel, uma prática chamada de "tempo de boxe." As pessoas encontram uma tentação natural para estender uma iteração quando a equipas cai para trás. Isto geralmente se mostrado uma estratégia ruim, pois leva extensões mais longas e mais longas para a iteração, colocando em risco o cronograma e desmotivar a equipas. Muitos gerentes bem-intencionadas danificar uma equipas ao estender a iteração indefinidamente, roubando a equipas da alegria e celebração em torno da conclusão.

Uma estratégia melhor é fixar a data final, e ter a equipas entregar o que eles tenham concluído no final da caixa de tempo. Com esta estratégia, a equipas aprende o que pode completar nesse período de tempo, feedback útil para o plano do projeto. Ela também fornece a equipas com um Victory cedo.

iterações de tamanho fixo permitir que a equipas para medir a sua a sua velocidade de movimento do projeto de *velocidade.* comprimentos fixos iterações dar esse ritmo ao projeto que as pessoas descrevem como o projeto "batimento cardíaco".

Algumas pessoas bloquear as necessidades durante uma caixa de iteração ou tempo. Isto dá a paz equipa de espírito enquanto elas se desenvolvem, assegurando-lhes que eles não terão de mudar de direção, mas pode completar *alguma coisa* finalmente. Uma vez eu encontrei um grupo experimentar XP aonde o cliente não queria que o julgamento para ter sucesso: este cliente mudou as prioridades requisitos cada poucos dias para que depois de várias iterações a equipas ainda não tinha conseguido completar qualquer história de um utilizador. Em tais ambientes hostis, tanto o bloqueio requisitos e a paz de espírito-requisitos críticos bloqueio raramente é necessária em ambientes bem-comportados.

Os resultados de uma iteração pode ou não pode ser liberado. Apenas quantas vezes o software deve ser enviado para os utilizadores reais é um tema para toda a equipas, incluindo o patrocinador, para deliberar. Eles podem achar que é prático para entregar depois de cada iteração, eles podem entregar a cada poucas iterações, ou podem coincidir com as entregas para datas específicas do calendário.

Entrega Frequent é sobre a entrega do software para os utilizadores, não apenas iteração. Uma equipas de projeto nervoso eu visitei havia sido iteração mensal por quase um ano, mas ainda não entregue qualquer lançamento. As pessoas estavam ficando muito nervoso, porque *o cliente não tinha visto o que tinham vindo a trabalhar sobre para o último ano!* Isto constitui uma violação da Entrega frequente.

Se a equipas não pode entregar o sistema para a base de utilizadores completo a cada poucos meses, *visões de utilizadores* tornam-se ainda mais crítica. A equipas precisa mandar para os utilizadores a visitar a equipas e ver o software em ação, ou pelo menos no utilizador para instalar e testar o software. A falha em manter estas visões de utilizadores facilmente se correlaciona para acabar com o fracasso do projeto, quando os utilizadores finalmente, e tarde demais, identificar que o software não atende às suas necessidades.

Para o melhor efeito, exercer tanto embalagens e implantação. Instalar o sistema em tão perto de uma situação mais real possível.

Propriedade 2. **Melhoria reflexivo**

A descoberta de que me pegou completamente de surpresa foi que um projeto pode reverter sua sorte de falha catastrófica para o sucesso se a equipas vai se reunir, listar o que tanto é e não está funcionando, discutir o que poderia funcionar melhor, *e fazer essas mudanças* na próxima iteração. Em outras palavras, refletir e melhorar. A equipas não tem que gastar uma grande quantidade de tempo fazendo este trabalho - uma hora a cada poucas semanas ou meses vai fazer. Só o fato de tomar o tempo fora do atabalhoadamente do desenvolvimento diária para pensar sobre o que poderia funcionar melhor já é eficaz.

Você se reunir pelo menos uma vez nos últimos três meses por uma meia hora, hora ou meio dia para comparar as notas, refletir, discutir hábitos de trabalho do seu grupo e veja o que acelera -lo, o que você fica mais lento, eo que você pode ser capaz de melhorar?

* * * *

O projeto que me deu a surpresa foi *Projeto Ingrid (* descrito em *Sobrevivendo Projetos OO).* No final da primeira iteração - que deveria ser de quatro meses de duração, mas eles tinha estendido - eles estavam muito atrasados, desmoralizados, e com o que eles reconhecido como um projeto inaceitável. Foi o que fez em seguida que me surpreendeu: Eles lançaram 23 dos 24 programadores do lado do cliente para voltar a seus antigos empregos, contratou 23 novas pessoas, mudaram as estruturas de equipas e de gestão, pagos por várias semanas de treinamento programador, e começou mais, exigindo o novo grupo para refazer o trabalho da primeira equipas e fazer progressos adicionais.

No final da segunda iteração, eles estavam atrás novamente agendar, mas tinha um projeto que iria realizar, e a estrutura da equipas e programadores estavam funcionando. Eles realizada outra oficina reflexão, realizou alterações adicionais, e continuou.

Quando entrevistei-los, eles estavam em sua quarta versão, antes do previsto e conteúdo com o seu design e as suas práticas de trabalho.

Desde que a entrevista, tenho notado que a maioria dos projetos que visitei teve um começo difícil, ou mesmo uma catástrofe. Isso é tão comum que eu têm vindo a esperar, quase mesmo recebê-lo: A partir desse primeiro catástrofe vêm todos os tipos de informações novas e importantes sobre o ambiente de trabalho do projeto, que seria mortal, mas escondido.

Em *projeto Winifred* conseguimos no final do primeiro ciclo de entrega de três meses que eu chamei um release "chiclete" (o sistema foi mal realizada em conjunto pelo equivalente de chiclete software). No entanto, entregamos algo a cada três meses, ficando melhor e melhor a cada vez, até que finalmente entregue a função contratada no tempo.

> **Após cada entrega, alguns de nós ficamos juntos. Nós identificamos o que não estava funcionando e discutiram formas de corrigi-lo. Mantivemos a tentar novas estratégias até encontrarmos aqueles que trabalharam. Entrega Frequent e Melhoria reflexivo tornou-se fatores críticos de sucesso para nós como eles são para tantos projetos.**

* * *

As pessoas, a tecnologia e a mudança de atribuição ao longo de um projeto. As convenções dos usos da equipas precisam mudar para corresponder.

As pessoas da equipas são a melhor equipados para dizer o que é mais adequado para sua situação, razão pela qual Crystal Clear deixa tantos detalhes não declarada, mas para a equipas para finalizar. o Melhoria reflexivo mecanismo
permite-lhes fazer esses ajustes.

A cada poucas semanas, uma vez por mês, ou duas vezes por ciclo de entrega, as pessoas se reúnem em um *oficina de reflexão* ou *retrospectiva iteração* para discutir como as coisas estão funcionando. Eles observam as convenções que se manterão e aqueles que querem alterar para o próximo período,
e colocaremos essas duas listas com destaque para os membros da equipas para ver durante o trabalho na próxima iteração.

Seja qual for a frequência, o formato de reuniões e técnica utilizada, equipas bem sucedidas manter esta discussão periodicamente e experimentar novas ideias. Selecção podem tentar, em várias formas: a programação em pares, de teste de unidade, orientada para o teste de desenvolvimento, quartos simples versus múltiplos assentos quarto, vários níveis de envolvimento do cliente, e mesmo diferentes comprimentos de iteração. Estas são todas as variações apropriados dentro Crystal Clear.

Para que as pessoas dizem que estão usando Crystal Clear, não é necessário que eles continuam a usar as convenções de partida. Na verdade, espera-se que eles vão tentar novas ideias. Em uma reunião do grupo de utilizador de cristal, as pessoas discutem o que tinha experimentado com, como se sentiam sobre essas experiências, e como eles evoluíram as suas convenções de trabalho. Uma equipas pode denunciar mover as reuniões de cada duas semanas para cada mês, um outro movimento do formato que eu descrevo no próximo capítulo a uma discussão em linha reta de valores das pessoas durante o desenvolvimento.

Eu gosto de usar o *workshop de reflexão* descrito no capítulo Técnicas. O livro de Norm Kerth, *Projeto Retrospectivas,* apresenta um formato estendido, juntamente com muitas atividades para tentar dentro da oficina. As especificidades do formato de workshop não são quase tão importante como o fato de que a equipas está segurando um.

Propriedade 3. **Comunicação osmótica**

Comunicação osmótica significa que a informação flui para o fundo audição dos membros da equipas, para que eles peguem a informação relevante como que por osmose. Isto é normalmente realizado por estar los na mesma sala. Então, quando uma pessoa faz uma pergunta, outros na sala pode entrar em sintonia ou sintonizar, contribuindo para a discussão ou continuar com o seu trabalho. Várias pessoas têm relacionado a sua experiência dele tanto quanto essa pessoa fez:

> **Nós tivemos quatro pessoas fazendo a programação em pares. O patrão entrou e perguntou o meu parceiro uma pergunta. I começou a responder, mas deu o nome errado de um módulo. Nancy, a programação com Neil, corrigiu-me, sem Neil nunca perceber que ela tinha falado ou que uma pergunta tivesse sido feita.**

Quando Comunicação osmótica está no lugar, perguntas e respostas fluir naturalmente e com surpreendentemente pouca perturbação entre a equipas.

Comunicação osmótica e Entrega Frequent facilitar o feedback tão rápida e rica que o projeto pode operar com muito pouca outra estrutura. É por isso que estas duas propriedades são os dois primeiros listados.

Você leva 30 segundos ou menos para obter a sua pergunta para os olhos ou ouvidos da pessoa que pode ter a resposta? Você ouvir algo relevante a partir de uma conversa entre outras os membros da equipas, pelo menos a cada poucos dias?

* * * *

Comunicação osmótica é a versão mais potente que os pequenos projetos pode atingir de Fechar Comunicação, um núcleo de propriedade para toda a família de Cristal. Comunicação osmótica faz com que o custo das comunicações de baixa e a taxa de retorno alta, para que os erros sejam corrigidos de forma extremamente rápida e o conhecimento é disseminado rapidamente. As pessoas aprendem as prioridades do projeto e que detém as informações. Eles pegar novas programação, design, teste e manuseio ferramenta truques. Eles pegar e corrigir pequenos erros antes que eles se transformar em outros maiores.

Apesar Comunicação osmótica é valiosa para projetos maiores, é, naturalmente, cada vez mais difícil de alcançar, como o tamanho da equipas cresce.

É **difícil simular Comunicação osmótica sem ter as pessoas na mesma sala, quartos adjacentes de duas ou três pessoas cada confere muitos dos benefícios. Herring (1999) relataram o uso de intranet de alta velocidade com câmeras web, microfones e sessões de bate-papo para trocar perguntas e código, para simular o quarto individual para (alguns) medida. Com boa tecnologia, as equipas podem obter conseguir alguma aproximação das Fechar Comunicação para alguns propósitos, mas eu ainda tenho que ver Comunicação osmótica**

conseguida com outra do que a proximidade física entre os membros da equipas.

* * *

Discussão de Comunicação osmótica conduz inevitavelmente a discussão sobre o layout de escritório e mobiliário de escritório.

Crystal Clear precisa de pessoas ser muito próximos uns dos outros para que eles ouvir informações úteis e obter perguntas respondidas rapidamente. A maneira óbvia de fazer isso é colocar todos em um quarto individual ("sala de guerra"), repetidamente mostrado como sendo muito eficaz (Radical Colocation ref ???).

Figura 2-1. Osmótica Comunicação (Graças a Tomax)

Muitas pessoas que têm escritórios privados resistem a mover-se em um espaço de grupo. No entanto, às vezes você pode transformar limões em limonada (por assim dizer) com este movimento:

> **Lise foi informado por sua gestão que seu departamento teria de reduzir o número de pés quadrados que eles usaram. Isto significava desistir de consultórios particulares. Ela sugeriu que seu povo trabalhar juntos e criar seus próprios espaços de escritório, de três a cinco pessoas em uma área combinada. Os grupos colocar menos pés quadrados ao redor de cada mesa de trabalho para que eles pudessem alocar espaço para áreas adicionais com cadeiras, sofá ou, em alguns casos, suas próprias salas de reuniões.**

As imagens seguintes mostram o que um grupo veio com. Note-se que apesar de terem menos de pés quadrados pessoa que antes, eles acabaram com linhas de visão mais longas e uma área de conversa com cadeiras macias.

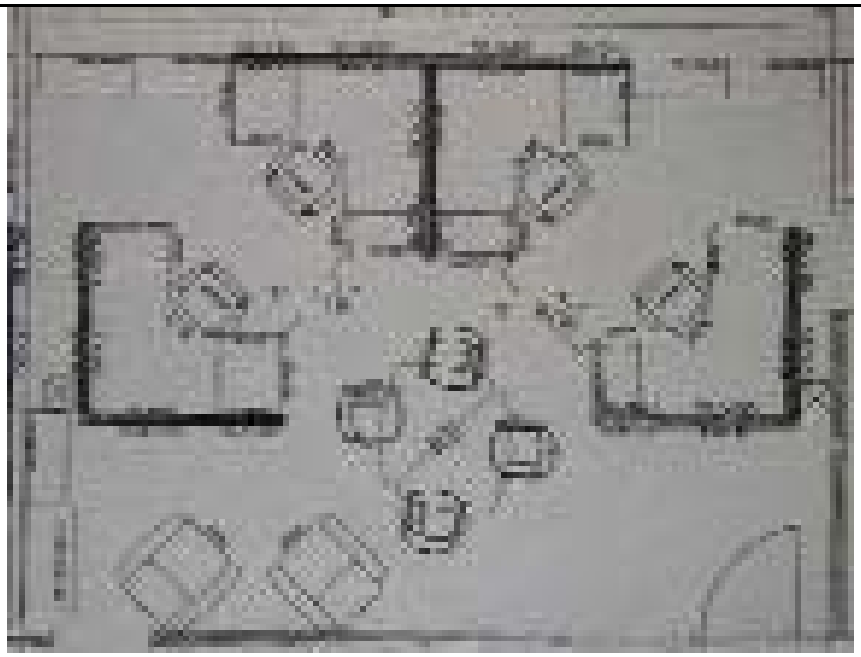


Figura 2-2. piso plano (Graças a Lise Hvatum)

Figura 2-3. Foto do mesmo escritório (Graças a Lise Hvatum)

Figura 2-4 mostra a sala um pequeno grupo colocar no lado de seu escritório compartilhado. Eles usaram para falar sem perturbar quem ainda estava programando, e também para deixar suas notas de design e planeja-se na parede.

Figura 2-4. sala de trabalho grupo ligado ao escritório compartilhado (Graças a Lise Hvatum) o grupo de Lise usou a habitual mobiliário de escritório: côncava, projetado para ter o CRT gordura volta para o canto. Este tipo de tabela apresenta uma desvantagem para uma equipas de desenvolvimento ágil, porque é difícil para uma segunda ou terceira pessoa a ver a tela. A sala de guerra na Figura 2-1 pode parecer menos glamourosa, mas existe um utilitário nessas tabelas feio: As pessoas podem se reunir em torno de uma tela, pares de pessoas podem trabalhar juntos facilmente. É por esta razão que as equipas de desenvolvimento ágeis preferem mesas retas, ou melhor ainda, tabelas essa protuberância para fora em direção a datilógrafa.

Se você configurar uma área de trabalho-sala de guerra, não se esqueça de arranjar outro lugar para as pessoas irem para relaxar e fazer o seu e-mail privado. Isto permite às pessoas para se concentrar quando pisam em área comum, e encontrar um pouco de alívio da pressão, ao sair. Tal arranjo é referido como um "cavernas e comum" arranjo.

Figura 2-5. área de discussão em grupo (graças a Darin Cummins)

Uma equipas de projeto tem permissão para criar um espaço comum de discussão com cadeiras macias e sofá (Figura 2-5). Na parede em frente às cadeiras é a sempre presente quadro com dispositivo de captura de quadro branco. Este é o lugar aonde a equipas suspensa para manter suas discussões Design Group, reuniões de planificação de iteração e oficinas de reflexão.

Desenvolvimento Ágil de Software (**Cockburn 2003) contém informações adicionais sobre "as correntes de convecção" do fluxo de informações dentro de um grupo, Osmótica Comunicação, o valor de colocation, e exemplos de layout do escritório.**

* * *

Comunicação osmótica gera seus próprios perigos, mais comumente ruído e um fluxo de perguntas para o desenvolvedor mais experiente da equipas. As pessoas geralmente auto-regular aqui, solicitar menos conversa fiada ou mais respeito pelo tempo de raciocínio.

A tentativa de "proteger" o designer-chefe com um escritório privado geralmente sai pela culatra. Essa pessoa realmente precisa ser sentado no meio da equipas de desenvolvimento. O designer-chefe é muitas vezes o especialista em tecnologia, especialista de domínio, e o melhor programador, e assim é necessariamente em alta demanda. Quando ela é tirado, os desenvolvedores mais jovens perca a chance de desenvolver bons hábitos de desenvolvimento, perca crescente no domínio e da tecnologia, e cometer erros que de outra forma seria pego muito rapidamente. O custo para o projeto acaba por ser é maior do que o benefício do tempo tranquilo para o Designer de chumbo. Ter o designer de chumbo na mesma sala que o resto da equipas é uma estratégia chamada Especialista em Earshot, descrito em (Cockburn URL-EIE). Especialista em Earshot

é um uso especial Comunicação osmótica.

Mesmo a melhor propriedade sucesso não é adequada em determinadas circunstâncias. Comunicação osmótica não

é exceção. Se o designer-chefe fica tão sobrecarregado e tão frequentemente interrompido por ser incapaz fazer progressos em qualquer coisa, ela precisa de um local de trabalho sem interrupções em tudo e comunicações extremamente limitado com a equipas, um Cone do Silêncio, Eu chamo-lhe. Muitos designers de chumbo usar as horas de 6:00 a 2

sou como seu Cone do Silêncio, mas é melhor para todos os envolvidos se um aceitável Cone do Silêncio pode ser configurado dentro de horas de trabalho normais. o Cone do Silêncio estratégia é descrita em detalhe em (Cockburn URL-COS).

Propriedade 4. **Segurança pessoal**

Segurança pessoal é ser capaz de falar quando algo está incomodando você, sem medo de represálias. Pode envolver dizendo ao gerente que o cronograma é irrealista, um colega que seu projeto precisa de melhorias, ou até mesmo deixar um colega sabe que ela precisa tomar um banho mais vezes. Segurança pessoal é importante porque com ele, a equipas pode descobrir e reparar suas fraquezas. Sem ele, as pessoas não vão falar, e as fraquezas vai continuar a prejudicar a equipas.

Segurança pessoal é um passo inicial na direção Confiar em. Trust, que envolve dar a alguém poder pessoa sobre si mesmo, com acompanhamento de risco de danos pessoais, é a medida em que um é confortável com entrega essa pessoa o poder. Algumas pessoas confiar nos outros por padrão, e esperar para ser ferido antes de retirar a confiança. Outros são inclinados a confiar nos outros, e esperar até que eles vêem evidência de que não será ferido antes que eles dão a confiança. Presença de confiança é positivamente correlacionada com o desempenho da equipas (Costa, 2002).

As diferentes maneiras em que se pode ser ferido levar a diferentes formas de confiança e desconfiança (Tyler, 1996). Uma pessoa carente honestidade aberta pode mentir ou esconder. Aquele que não tem congruência em ações serão inconsistentes. Uma pessoa sem qualquer competência ou confiabilidade deixará de comple atribuições. Uma preocupação pessoa faltando para outros podem agir para danificá-los, inclusive dando informações confidenciais.

Aceitando exposição a teses variaram potenciais danos utiliza diferentes formas de confiança. É realista nem necessário pedir a todos em um projeto para confiar uns nos outros em todos eles. É importante que as pessoas podem falar e agir livremente - eles precisam confiar uns nos outros no que diz respeito a ações prejudiciais e traição.

Quando uma pessoa vê que os outros não vão trair ou prejudicar a sua base nas informações que ela revela, ela irá revelar informações mais livremente, o que irá acelerar o projeto. Portanto, Segurança pessoal é a propriedade fundamental para alcançar.

você pode dizer ao seu chefe que você mis-estima em mais de 50%, ou que você acabou de receber uma oferta de trabalho tentadora? você pode não concordar com ela sobre a programação em uma reunião de equipas? pode

as pessoas acabam longos debates sobre projetos uns dos outros com discordância amigável?

* * * *

Estabelecer a confiança envolve estar em uma situação aonde um desses perigos está presente, e vendo que as outras pessoas não te machucar. Em outras palavras, para construir a confiança, deve haver exposição.

Três exposições particulares são relevantes no desenvolvimento de software:

- **revelando sua ignorância,**
- **revelando um erro, e**

- **revelando sua incapacidade em uma atribuição.**

Os líderes hábeis expor os seus membros da equipas (e a si mesmos!) para estas situações cedo, e, em seguida, demonstrar com velocidade e autenticidade que não só vai prejudicar não acumulam, mas que o líder e da equipas como um todo vai agir para apoiar a pessoa.

> **O responsável pelo projeto 11 me disse que quando uma nova pessoa se juntou a sua equipas, ela iria visitar essa pessoa em particular para discutir seu trabalho e progresso, e esperar o momento inevitável quando ele teve que admitir que ele não tinha feito ou não sabia alguma coisa.**
>
> **Este foi o momento crucial para ela, porque até ele revelou uma fraqueza, ela não podia demonstrar-lhe que ela iria cobrir para ele ou levá-lo assistência. Ela sabia que não estava indo para obter tanto informações confiáveis e plena cooperação com ele até que ele entendeu corretamente que quando ele revelou uma fraqueza ou erro, ele seria realmente obter assistência. Ela disse que algumas pessoas entenderam a mensagem depois de sua primeira visita, enquanto outros necessários várias manifestações antes de abrir.**

Outro líder do projeto disse de construção da coesão e segurança na equipas por ter o trabalho de grupo em conjunto para resolver um problema difícil que eles estavam enfrentando. Ao resolver o problema juntos, eles aprenderam várias coisas:

- **Primeiro, eles não iria se machucar se eles admitiram ignorância, mesmo em sua própria área.**
- **Em segundo lugar, eles aprenderam como interpretar maneirismos uns dos outros como não-ameaçador, mesmo na argumentação pesada.**
- **Finalmente, eles aprenderam que juntos eles poderiam resolver coisas que não poderia resolver sozinho.**

A confiança é reforçada com Entrega frequente. Quando o software é entregue, as pessoas reconhecem que fez a sua parte do trabalho e que se esquivou, que disse a verdade, que danificado ou protegido contra quem, e que, apesar de suas maneiras superficiais, pode ser confiável ao longo do qual as dimensões. Com Segurança pessoal, eles falam de seu coração durante o Melhoria reflexivo sessões.

* * *

Segurança pessoal anda de mãos dadas com *amicability,* o wilingness a ouvir com boa vontade. O projeto sofre quando qualquer pessoa na equipas pára escutando com boa vontade, ou perde a inclinação para repassar informações possivelmente importante. Além de habilidade pessoal, o progresso para a frente de um projeto depende apenas da velocidade de circulação da informação através de pessoas ("Meme-metros por minuto", se você preferir).

Normalmente, uma pessoa da equipas define a liderança na amicability. Em um projeto maior, muitas vezes é crucialmente o gerente de projeto. Em um projeto Crystal Clear, ele pode ser qualquer um na equipas. A menos que haja uma razão específica contrariar isso, amicability se espalha rapidamente e faz com que a equipas mais confortável em trocar informações rapidamente. Segurança pessoal

---

11 Graças a Victoria Einarsson, na Suécia.

e amicability juntos ajudar a levar a Colaboração através das fronteiras organizacionais, o estabelecimento de linhas de vida globais para o projeto. I definir amicability como elemento de gestão significativa em um projeto, em parte como evidência para Segurança pessoal.

Uma vez Segurança pessoal e amicability são estabelecidas, pode surgir um instrumento útil, dinâmica lúdica. As pessoas podem travar a concorrência uns com os outros. Eles podem discutir em voz alta, mesmo à beira da luta, sem levá-la pessoalmente. No caso em que alguém não levá-la pessoalmente, eles classificar e acertar as coisas novamente.

Tenha cuidado, porém, para não confundir Segurança pessoal com polidez. Algumas equipas parecem ter Segurança pessoal no lugar, mas na verdade estão apenas sendo educado, porque eles não estão dispostos a mostrar desacordo. 12 Cobrindo as suas divergências com polidez e conciliação, eles não detectar e corrigir erros que estão presentes. Isso prejudica o projeto, no final, como no caso de sobre-amicability descrito em (Cockburn 2002, pp. ???).

Há uma boa quantidade de literatura sobre o assunto de confiança, alguns dos quais você pode achar aplicáveis à sua situação. Leia mais em Hohmann (1997, pp. 250 ???), Karmer (1996), Costa, e Adams (URL).

12 Graças a Kay Johanssen para esta distinção.

Propriedade 5. **Foco**

Foco é primeiro saber o que trabalhar, e, em seguida, ter tempo e paz de espírito para trabalhar nele. Saber o que trabalhar em vem de comunicação sobre a direção e prioridades objetivo, normalmente a partir do *Patrocinador executivo.* Tempo e paz de espírito vêm de um ambiente aonde as pessoas não são tomadas longe de sua tarefa para trabalhar em outros, coisas incompatíveis.

Não todas as pessoas sabem o que seus dois principais itens prioritários para trabalhar em Are? eles são garantidos pelo menos dois dias em uma fila e duas horas ininterruptas a cada dia para trabalhar com eles?

* * * *

Mesmo com a melhor das intenções, os programadores vão trabalhar em coisas que trazem única aleatoriamente valor do negócio se eles não dizem o que irá fornecer valor de negócios. É o trabalho do *Patrocinador executivo,* a partir do projeto *fretamento* atividade e funcionando de forma contínua ao longo do projeto, para torná-lo claro para todos, aonde as prioridades da organização mentir.

O vice-presidente de uma empresa de 50 pessoas se sentou uma noite, priorizou os 70 pendentes iniciativas da empresa, e anunciou os resultados para os gestores no dia seguinte. Ela foi em torno de cada desenvolvedor e fez com que cada um deles conhecia os dois principais itens para eles, individualmente.

1 *Designer-chefe* Eu conheci manteve declaração de missão do projeto e prioridades afixado na parede e se referiu a eles regularmente.

Basta saber o que é importante não é o suficiente. Desenvolvedores informar regularmente que as reuniões, solicitações para dar demos, e pede para corrigir erros em tempo de execução mantê-los de completar seu trabalho. É bastante normalmente leva uma pessoa cerca de 20 minutos e energia mental considerável para recuperar a sua linha de pensamento após uma dessas interrupções. Quando as interrupções acontecem três ou quatro vezes por dia, não é incomum para a pessoa para a marcha lenta simplesmente entre interrupções, sentindo que não vale a pena a energia para obter profundamente em uma linha de pensamento quando a próxima distração só vai aparecer no meio dela.

Pessoas pediram para trabalhar em dois ou três projetos ao mesmo tempo informar regularmente que eles são incapazes de fazer progresso em qualquer projeto. Parece demorar uma hora e meia para uma pessoa a recuperar a sua linha de pensamento, depois de trabalhar em um projeto diferente.

Entre os gerentes de projeto experientes que eu entrevista, o consenso é que os projetos sobre um ano e meio é o máximo que uma pessoa pode estar ligado e permanecer eficaz. No momento em que um terceiro projeto é adicionado, o desenvolvedor torna-se ineficaz em todos os três. Compare isso com os gerentes inexperientes que, subestimando a custos de transferência entre projetos, atribuir desenvolvedores para trabalhar em três a cinco projetos no

mesmo tempo. Eu encontrei um desenvolvedor atribuído a 17 projetos em simultâneo! Você pode imaginar que ele mal teve tempo para relatar nas diversas reuniões sua contínua falta de progresso em todas as frentes.

O reparo é simples, embora desconfortável. o *Patrocinador* deixa claro que projetos e itens de trabalho são prioridade para cada pessoa, e organiza para os dois primeiros itens a ser nitidamente superior na prioridade do que todo o resto.

A equipas deve, então, adotar convenções que fornecem Foco tempo para os membros da equipas. Uma dessas é a convenção que quando uma pessoa começa a trabalhar em um projeto, ela é garantida, pelo menos, dois dias antes de ter que mudar para um segundo projeto. Isto permite alguma mudança de projeto, assegurando simultaneamente o tempo suficiente pessoa a fazer progresso real em vez de usar o tempo todo só para voltar até a velocidade em cada projeto antes de sair novamente.

A próxima convenção adotar pode ser para localizar interrupções que distraem. A minha experiência é que é geralmente impraticável para engarrafar interrupções para algo tão limpo e arrumado como "única manhã" ou "entre 1 e 3 horas da tarde." É da natureza de interrupções para vir esporadicamente e com alta prioridade. O que a equipas pode fazer

é **criar uma janela de tempo de duas horas durante as quais interrupções estão bloqueadas. Há muito poucas interrupções que não podem esperar por duas horas. Algumas equipas usam a partir das 10:00 ao meio-dia como um momento em reuniões, telefonemas e demos não são permitidos.**

Com duas horas de tempo de foco garantida a cada dia, e dois dias seguidos no mesmo projeto, um desenvolvedor que de outra forma está sendo conduzida à distracção pode obter quatro horas completas do trabalho realizado em uma semana. Um desenvolvedor que adoptou estas relatado depois de algumas semanas que ele tinha obtido mais feito naquelas poucas semanas do que nos vários meses antes disso.

Propriedade 6. **Fácil acesso a utilizadores experientes**

Acesso continuou a utilizador (s) especialista fornece a equipas com

- **um lugar para implantar e testar o As entregas frequentes,**
- **feedback rápido sobre a qualidade do seu produto acabado,**
- **feedback rápido em suas decisões de design e**
- **up-to-date requisitos.**

Pesquisadores Keil e Carmelo publicou resultados que mostram o quão crítico é ter links diretos para utilizadores experientes (Keil 95). gerentes de Topografia que haviam trabalhado com e sem acesso fácil para os utilizadores reais, eles escrevem:

> **"..., Em 11 dos 14 casos emparelhados, o projeto mais sucesso envolveu um maior número de ligações que o projeto menos bem sucedida.... Esta diferença foi estatisticamente significativa em um teste t emparelhado ($p < 0,01$)."**

Sua pesquisa levou-os a uma recomendação específica: "reduzir a dependência de links indiretos" Em outras palavras, tenha acesso real para os utilizadores reais.

Leva menos de três dias, em média, a partir de quando você chegar a uma pergunta sobre o uso do sistema quando um utilizador experiente responde a pergunta? você pode obter a resposta em um algumas horas?

* * * *

Tudo muito bonito, mas quantos utilizadores, e quanto tempo?

Mesmo uma hora por semana de acesso a um utilizador real e especialista é imensamente valioso. Quanto mais horas por semana que um utilizador experiente está disponível para uma equipas, mais vantagem que pode tirar dessa proximidade. A primeira hora, no entanto, é o mais crucial.

A outra coisa que é importante é o período de tempo até que uma pergunta é respondida. Se uma pergunta não será respondida por mais três dias, os programadores são susceptíveis de colocar no código de sua melhor suposição atual, e pode esquecer de verificar novamente a sua decisão quando estão com os utilizadores novamente. Portanto, eles devem ter acesso de telefone para o utilizador especialista durante a semana.

Aqui estão os métodos de acesso de três utilizadores que ouço com mais frequência:

- ***encontros de utilizadores semanais ou semi-semanais com telefonemas adicionais.*** **Você pode achar que o utilizador carrega a equipas com informações nas primeiras semanas. Com o tempo, os desenvolvedores precisam de menos tempo do utilizador (s), como elas se desenvolvem código. Eventualmente, se forma um ritmo natural, como o utilizador fornece novas exigências de informação e também analisa projeto de software. Este ritmo natural pode envolver uma, duas ou três horas por semana por utilizador especialista. Se você adicionar alguns telefonemas durante a semana, então perguntas são respondidas com rapidez suficiente para manter a equipas de desenvolvimento de ir fora em um falso sentido.**

- ***Um ou mais utilizadores experientes diretamente na equipas de desenvolvimento.*** **Esta é apenas raramente é possível, mas não descontá-lo. I periodicamente encontrar uma equipa localizado dentro da comunidade de utilizadores, ou seja, de alguma forma colocado com um utilizador experiente.**
- ***Enviar os desenvolvedores para se tornar utilizadores estagiários por um período.*** **Odd embora isso possa parecer, algumas equipas de desenvolvimento de enviar os desenvolvedores a utilizadores de sombra ou os próprios utilizadores aprendizes se tornam. Enquanto eu não tenho uma base muito grande de histórias para retirar, eu ainda não ouvi uma história negativa relacionada a usar esta estratégia. Os desenvolvedores voltar com um apreço e respeito para os utilizadores, e um apreço pelo modo como seu novo software pode mudar as vidas de trabalho dos utilizadores.**

Keil e Carmelo nome ligações de utilizadores adicionais, incluindo equipas facilitadas, prototipagem de interface pelo utilizador, entrevistas, testes, quadros de avisos, laboratórios de usabilidade, estudo observacional, e grupos focais. Em uma rápida pesquisa na internet, eu transformou-se um número de companhias que se especializam em encontrar temas e teste de software com os utilizadores reais.

I distinguir entre o especialista *do utilizador* e a *o negócio* perito, porque muitas vezes são pessoas diferentes. O especialista em negócios conhece as políticas de negócios, incluindo que são fixados, que são propensos a mudar e as dependências entre eles. Os utilizadores geralmente não sabe esta informação. Por outro lado, o utilizador experiente sabe que as operações são comuns e que são raros, o que atalhos são necessários, que informação realmente não tem que ser inserido, e as informações que precisam estar visíveis ao mesmo tempo. O especialista em negócios não vai saber esta informação, uma vez que se trata apenas de operação diária contínua.

A equipas de desenvolvimento irá conter uma *Business Expert (* consulte Funções, no Capítulo 5). Essa pessoa pode ser o patrocinador, ou o utilizador especialista, ou pode ser o Designer de chumbo. Tal pessoa é quase sempre disponível para um projeto, e então eu não mexer tanto sobre isso. o

Utilizador Expert, **por outro lado, é geralmente ausente, em detrimento do projeto, que é por isso que eu mexer tanto sobre isso aqui. Fácil acesso a utilizadores experientes fornece uma rede de segurança para a equipas, além de ser uma vantagem competitiva. É provável que seja um fator crítico de sucesso para uma equipas pequena.**

* * * *

"OK, nós temos os utilizadores -? Agora o que vamos fazer com eles" Você precisa saber o que eles querem, o que os seus patrocinadores estão dispostos a pagar, aonde seus padrões de uso rápido e raro-mas-significativas mentir, quer você tenha esquecido algo crítico. Você precisa dos utilizadores antes, durante e depois da concepção.

Antes **você chegar muito longe na concepção do sistema, você precisa identificar as funções de utilizador que os patrocinadores considerar as pessoas mais importantes para se ajustar à aplicação. Estes são o *papéis focal.* O sistema irá apresentar diferentes "personalidades" (rápido e eficiente, por exemplo, ou quente e amigável) para cada função diferente. Os designers vai acentuar**

uma personalidade sobre os outros, e você quer ter certeza de que eles acentuam o mais importante (s).

A técnica descrita no capítulo seguinte, *Interaction Design Essencial,* é uma maneira de identificar os papéis e personalidades para desenvolver focais. A atração desta técnica oficina é que você pode reunir as informações em apenas alguns dias.

***Durante* design, você precisa de respostas para muitas pequenas questões. Para isso você precisa Fácil acesso a utilizadores experientes numa base contínua, como descrito nesta secção.**

***Depois de* design, quando você pensa que você é feito, você precisa de utilizadores novamente, para avaliar seus resultados. Se o sistema irá para alguns utilizadores, locais, simplesmente convidá-los para um test drive. Se, por outro lado, você tem um grande número de utilizadores geograficamente dispersos, então o custo da avaliação é maior. Eu não sei de nenhum eficiências especiais para esta situação. Técnicas para avaliação de usabilidade foram descritos há décadas (grupos de foco no cliente e amostras de usabilidade, sendo os principais exemplos). Eu descobri recentemente, há toda uma indústria para testes de usabilidade 13.**

Antes de deixar esta propriedade, peço-lhe que leia novamente os últimos parágrafos do Entrega frequente, em que eu descrevo os problemas decorrentes da não organizando para *real*

feedback do utilizador. Mesmo equipas que fazem todas as outras práticas de desenvolvimento ágil, se encontram diante catastrófica má notícia no final do projeto se eles negligenciam esse feedback durante o projeto.

13 No Google, por exemplo, ver Computadores> Interação Humano-Computador> Empresas e Consultores> Teste de Usabilidade.

Propriedade 7. **Ambiente técnica com testes automatizados,**

Gestão de Configuração e Integração Frequent

Os elementos que destacam nesta propriedade são tais elementos centrais bem estabelecido que é embaraçoso ter de mencioná-los em tudo. Vamos considerá-los um de cada vez e todos juntos.

Testes automatizados. Equipas que entregar com sucesso usando testes manuais, de modo que este não pode ser considerado um *crítico* fator de sucesso. No entanto, cada programador que entrevistei, que uma vez mudou-se para testes automatizados jurou *Nunca trabalhar sem eles novamente.* Eu acho isso nada menos do que surpreendente.

A razão tem a ver com a melhoria da qualidade de vida. Durante a semana, eles revisar seções de código sabendo que eles podem rapidamente verificar que não tinham inadvertidamente quebrado algo ao longo do caminho. Quando eles chegam código de trabalho na sexta-feira, eles vão para casa sabendo que eles serão capazes de segunda-feira para detectar se alguém tinha quebrado no fim de semana - eles simplesmente executar novamente os testes na segunda-feira de manhã. Os testes de dar-lhes a liberdade de movimento durante o dia e paz de espírito durante a noite.

Gestão de configurações. O sistema de gestão de configuração permite que as pessoas a verificar em seu trabalho de forma assíncrona, de volta muda para fora, encerrar uma configuração específica para o lançamento, e reverter para essa configuração mais tarde, quando o problema surge. Ele permite que os desenvolvedores desenvolver o seu código de ambos *separadamente* e *juntos.* Ele é constantemente citado por equipas como sua ferramenta mais importante não-compilador.

Integração frequente. Muitas equipas de integrar o sistema várias vezes por dia. Se eles não podem lidar com isso, eles fazem isso diariamente, ou na pior das hipóteses, a cada dois dias. Quanto mais frequentemente eles se integram, mais rapidamente eles detectam erros, os menos erros adicionais que se acumulam, o mais fresco seus pensamentos, e quanto menor a região de código que tem de ser procurado para a falta de comunicação.

As melhores equipas combinar os três em Integração-com-Test contínua. Eles pegar erros de nível de integração dentro de minutos.

você pode executar os testes do sistema para a conclusão sem ter que estar fisicamente presente? Será que todos os seus desenvolvedores verificar o seu código no sistema de gestão de configuração?

Será que eles colocar em uma nota útil sobre ele como eles check-in?

É o sistema integrado, pelo menos duas vezes por semana?

* * * *

Como frequentes devem Integração Frequent estar? Não existe uma resposta fixa para isso mais do que a questão de quanto tempo uma iteração de desenvolvimento deve ser.

Um designer-chefe informou-me que ele era incapaz de convencer ninguém em sua equipas para executar a construção de mais de três vezes por semana. Enquanto ele não encontrar este confortável, ele trabalhou para esse projeto. A equipas usou um mês longas iterações, teve Osmótica Communications, Melhoria reflexivo, Gestão de Configuração e alguns Teste automatizado no lugar. Tendo essas propriedades em lugar feito a frequência da sua Integração Frequent menos crítica.

As equipas mais avançadas usar uma máquina de build-e-teste, como Cruise Control 14 integrar e nonstop teste (nota:...! tendo esta máquina em funcionamento ainda não é suficiente os desenvolvedores têm que realmente verificar em seu código para a principal base de código de linha várias vezes por dia). Os postos de máquina os resultados do teste para uma página web que os membros da equipas deixam aberta em suas telas em todos os momentos. Uma equipas de desenvolvimento distribuído internacionalmente (obviamente não usando Crystal Clear!) Relata que esse uso de Cruise Control permite que os desenvolvedores manter a par da base de código mudando, que em certa medida atenua o seu ser em fusos horários diferentes.

Experimentar com frequência integração diferente, e encontrar o ritmo que funciona para a sua equipa. Incluir este tema como parte de sua Melhoria reflexiva. Para saber mais sobre o gestão de configuração, eu encaminhá-lo para *Princípios de Gestão de configuração e Prática (* Hass 2002) *Padrões de Gestão de Configuração (* Appleton e Berczuk 2002) e *Pragmatic Version Control usando CVS* pela "Pragmatic Programmers" (Thomas 2003). Você pode precisar de contratar um consultor para entrar por alguns dias, ajudar a configurar o sistema de gestão de configuração e tutor a equipas sobre como usá-lo.

* * *

Teste automatizado significa que a pessoa pode começar os testes em execução, vá embora, não ter que intervir ou olhar para as telas, e depois voltar para encontrar os resultados do teste de espera. Sem olhos humanos e sem dedos são necessários no processo. suites de teste de cada pessoa podem ser combinados em um muito grande que pode, se necessário, ser executado no fim de semana (ainda não necessitando de olhos humanos ou dedos).

Três questões surgem imediatamente sobre Automated Testing:

- Em que nível eles deveriam ser escritos?
- Como automatizado que eles têm de ser?
- A rapidez com que deve ser executado? além de *testes de usabilidade,* que são melhor realizadas por pessoas fora do projeto 15, Acho três níveis de testes acaloradamente discutidos:

14 www.CruiseControl. ???

15 Google ainda tem uma categoria para ele: Computadores> Interação Humano-Computador> Empresas e Consultores> Teste de Usabilidade.

- ***testes de aceitação orientados para o cliente em execução na frente do GUI* e baseando-se os movimentos do rato e do teclado;**
- ***testes de aceitação orientados para o cliente que executam apenas atrás da GUI,* testando as acções do sistema sem a necessidade de um rato ou teclado simulador; e**
- ***testes orientada-programador função, classe e um módulo (* comumente chamado *testes de unidade).***

o testes automatizados que meus entrevistados estão tão entusiasmados sobre são do último duas dessas categorias. testes de unidade Automatização de permitir que os programadores para verificar se o seu código não foi quebrada acidentalmente para fora sob-los enquanto eles estão adicionando novo código ou melhorar código antigo (*refatoramento).* Os testes de aceitação sem GUI fazer o mesmo para o sistema integrado, e são estáveis ao longo de muitas mudanças no design interno do sistema. Embora os testes de aceitação sem GUI são altamente recomendados, eu raramente encontrar equipas de usá-los, a razão para que eles exigem a arquitetura do sistema de separar cuidadosamente o GUI da função. Esta é uma separação que tem sido recomendado há décadas, mas poucas equipas gerenciar.

testes de sistema baseados na GUI automatizados não estão na lista curta altamente recomendado porque eles são caros para automatizar e deve ser reconstruída a cada mudança da GUI. Esta dificuldade torna ainda mais importante que a equipas de desenvolvimento cria uma arquitetura que suporta testes de aceitação sem GUI.

testes de unidade de um programador precisa para executar em segundos, não minutos. Correndo tão rápido, ela não vai perder a sua concentração enquanto são executados, o que significa que ela vai realmente se preocupam em executar os testes como ela funciona. Se os testes demorar alguns minutos para ser executado, ela é improvável que volte a executar os testes depois de digitar em apenas algumas linhas de código novo ou mover duas linhas de código para uma nova função ou classe.

Os testes podem demorar mais tempo quando ela verifica seu código para o Gestão de configurações sistema. Neste ponto, ela completou uma sequência de ações de design, e pode dar ao luxo de se afastar por alguns minutos, enquanto os testes executados.

Os testes de aceitação pode levar um longo tempo para executar, se necessário. Eu escrevo esta frase deliberadamente: a razão os testes executar um longo tempo deve ser porque há tantos testes ou se houver uma seqüência temporal complicado envolvidos, não porque o equipamento de teste é desleixado. Mais uma vez, se os testes executados rapidamente, eles vão se executar mais vezes. Para alguns sistemas, no entanto, os testes de aceitação precisa executar no fim de semana.

O Crystal Clear não mandato quando os testes são escritos. Tradicionalmente, os programadores e testadores escrever os testes após o código é escrito. Além disso, tradicionalmente, eles não têm muita energia para escrever testes depois de escrever código. Parcialmente por esta razão, mais e mais desenvolvedores estão adotando o desenvolvimento orientado a testes (Beck 200 ???).

A melhor maneira que eu conheço para começar com Teste automatizado é baixar uma cópia de idioma específico do X- *unidade* framework de teste (aonde X é substituído pelo nome do idioma), inventado por Kent Beck. Há sim *JUnit* para programadores Java, *CppUnit*
para programadores C ++, e assim por diante para o Visual Basic, Scheme, C, e até mesmo PHP. Em seguida, obter

Teste-Driven projeto (**Beck 2002 ???) ou Astel ?? ((Astel 2003? F?) E trabalhar através dos exemplos. A pesquisa na web irá transformar-se mais recursos em *X-unidade.***

Ambos *HttpUnit* e FIT de Ward Cunningham (Quadro de testes integrados) ajudar com testes de aceitação sem GUI. O primeiro é para testar fluxos HTML de sistemas web-based, este último para permitir que o especialista em negócios para criar seus próprios conjuntos de testes sem precisar saber sobre a programação. Robert Martin integrado FIT com a tecnologia wiki de Ward para criar Fitnesse 16. Muitas equipas usam planilhas para permitir que os especialistas em negócios para digitar facilmente em dados de cenário para estes testes de função do sistema.

Há, infelizmente, há bons livros sobre projetar o sistema para facilitar a aceitação GUI-menos testando (Plumtree 2004 ???). O Mac fez a idéia de interfaces programáveis mainstream para um tempo curto17, e scripting é padrão com o Microsoft Office. Em geral, no entanto, a prática tem submersa e é usado por um número relativamente pequeno de desenvolvedores pendentes. As poucas pessoas que conheço que poderia escrever esses livros são de programação muito ocupado.

* * *

Termino esta seção com uma pequena depoimento para o desenvolvimento orientado a testes que eu espero vai influenciar um ou dois leitores. Graças a David Brady para esta nota:

> **Ontem eu escrevi uma função que leva um argumento variável, como printf (). Essa função decompõe os argumentos da lista, e deixa toda a confusão em um ponteiro de função. O ponteiro aponta para uma função de cada objeto coletor de mensagem do console ou de um lado do kernel buffer de memória mensagem de pia objeto. (Este é apenas herança básico, mas é tudo Gooky porque eu estou escrevendo-lo em C.)**
>
> **De qualquer forma, no passado, eu esperaria um problema de que a complexidade para me parar por um período indefinido de tempo, enquanto eu tentei depurar todas as coisas bizarras e fascinantes que podem dar errado com uma configuração semelhante.**
>
> **Levei menos de uma hora para escrever o teste eo código utilizando o teste-primeiro. Meu teste foi muito simples, mas chegando com que foi provavelmente a parte mais difícil de todo o processo. Eu finalmente decidi que se a minha função retornou o número correto de caracteres escritos (o mesmo que printf), que eu inferir que a função estava funcionando.**
>
> **Com o teste no lugar, eu tinha uma quantidade incrível de foco. Eu sabia o que tinha de fazer o código de fazer, e não havia necessidade para passear sem rumo no código tentando apoiar todos os casos possíveis. Não, era apenas "obter este teste para executar". Quando eu tive o teste de corrida, fiquei surpreso ao perceber que eu estava realmente terminado. Não havia nada extra para adicionar; Eu estava realmente feito!**

---

16 www.fit.org ??? e www.fitnesse.org ??? respectivamente.

17 Veja http://www.mactech.com/articles/develop/issue_24/according.html para um artigo de Cal Simone.

Eu costumo cortar 350-400 linhas de código de produção de grau em um bom dia. Ontem eu não sinto como se eu tivesse um dia particularmente bom, mas eu cortei 184 LOC teste e 529 LOC produção, Ploc que eu * sei obras *, porque os testes me dizer isso, Ploc que inclui um dos 10 top-mais complicada coisas que eu já feito em C (que passou de "idéia" para "totalmente funcional" em menos de 60 minutos).

Uau. Estou vendido.

infecção teste. Dê-lhe um lugar quente e úmido para começar, e ele vai fazer o resto .... David Brady

Evidência: A colaboração através das fronteiras organizacionais

Há um efeito colateral de assistir a Segurança pessoal, *cordialidade* dentro da equipas, e Fácil acesso a utilizadores experientes: torna-se natural para incluir outras partes interessadas no projeto, bem.

Géry Derbier, trabalhando com o serviço postal francês (La Poste) para construir software para executar uma nova instalação para lidar com todo o correio vai para dentro e fora do norte da França, informou sobre seu uso de Cristal. Com 25 pessoas, a sua foi um projeto na categoria cristal amarelo. No entanto, ele sabia que os princípios da metodologias família de Cristal, em particular a "esticar para caber" princípio, e, portanto, escolheu para estender Crystal Clear à sua maior configuração sempre que possível.

> **Discutimos o seu projeto, e em um ponto coberto ligação do seu projeto para a equipas de testes de integração localizado a 30 km de distância e ao especialista em negócios e uso que trabalha para La Poste. Fiz perguntas do tipo: "Quantas vezes essa pessoa visite o time Como ele se sente sobre isso Como foi seu empresário se sente sobre o seu vindo tantas vezes???" As respostas de Géry foram, para ambos os grupos externos: "Um dia por semana; confortável, feliz por estar envolvido tão cedo."**

Depois de nossa discussão, percebi que Géry tinha construído a segurança adicional para seu projeto de Colaboração através das fronteiras organizacionais. Seu projeto foi feliz ligada em ambos os ambientes de clientes e integração com um colega em cada extremidade. O contrato da La Poste medidos e pagos de acordo com os resultados dos testes integrados a cada poucos meses (o Entrega frequente). Os executivos La Postar tem software entregue em incrementos crescentes e pagos em conformidade. chefes de Géry, que não tinham experiência anterior com entrega incremental, estavam felizes com isso também, uma vez que viu sua vez entrega regular em pagamentos regulares. Géry tinha uma estrutura de apoio em todos os lados.

Colaboração através das fronteiras organizacionais não é um dado resultado em qualquer projeto. É o resultado de trabalhar com amicability e integridade honestidade dentro e fora da equipas. É difícil de alcançar se a equipa não ter em si Segurança pessoal
e, em menor extensão, Entrega frequente. Eu considero a presença de uma boa colaboração entre fronteiras organizacionais como evidência parcial que alguns dos principais e sete propriedades de segurança estão a ser alcançados.

Reflexão sobre as propriedades

Eu não acredito que quaisquer procedimentos prescritos existe que pode assegurar que os projetos de terra na zona de segurança de cada vez. Nem, com exceção de desenvolvimento incremental, posso mostrar-se em um projeto com qualquer conjunto particular de regras na mão, apesar de eu ter meus favoritos. É por isso que Crystal Clear é construído propriedades ao redor críticas em vez de especificação de procedimentos.

Uma equipas de cristal trabalha para definir os sete propriedades no lugar, usando qualquer grupo convenções, técnicas e padrões caber sua situação. As convenções podem variar por projeto e por mês. Novas técnicas são inventados com cada nova tecnologia (e geralmente sai de moda novamente alguns anos mais tarde). Estes sete propriedades, por outro lado, foram aplicadas com bons projetos por décadas.

Minha intenção com Crystal é não invadir o funcionamento natural dos indivíduos no projeto sempre que possível, e para permitir que o máximo possível de variação entre diferentes equipas, enquanto ainda recebendo os diversos projetos para a zona de segurança. Para permitir variação, I deve remover restrições. Remoção de restrições significa encontrar mecanismos mais amplos que oferecem uma rede de segurança. Os que eu optar por confiar em são estes:

- **As pessoas são por natureza bons em olhar em volta e se comunicar.**
- **Eles tomam a iniciativa quando fornecido com informações.**
- **Eles fazem melhor em um ambiente que é seguro no que diz respeito à segurança emocional pessoal, e em particular a liberdade de ataques pessoais.**
- **Eles fazem o seu melhor trabalho, se eles podem satisfazer sua necessidade de contribuição, realização e**

orgulho-in-trabalho. A rede de segurança Crystal Clear é construído sobre essas coisas. Segurança pessoal dá às pessoas a coragem pessoal para compartilhar tudo o que descobrir. Comunicação osmótica dá-lhes a maior chance

para descobrir informações importantes do outro, e faz isso com muito baixo custo de comunicação. Melhoria reflexivo dá-lhes um canal para aplicar feedback para seu processo de trabalho. Fácil acesso a utilizadores experientes dá-lhes a oportunidade de descobrir rapidamente as informações relevantes a partir do utilizador (s). Entrega Frequent cria

feedback aos requisitos do sistema e do processo de desenvolvimento. O ambiente de desenvolvimento técnico, incluindo Testes automatizados, Gestão de Configuração e Integração Frequent permite que as pessoas para fazer com segurança mudanças no sistema. sincronizar as múltiplas mentes que estão em movimento, ao mesmo tempo, e obter feedback sobre estágios intermediários do sistema rapidamente. Foco permite que a equipas de gastar seu bem energia nas coisas mais importantes.

Ron Jeffries, uma vez caracterizada Crystal Clear como "Trazer alguns desenvolvedores juntos em paz, amor e harmonia, código de envio a cada dois meses, e bom software vai surgir." Ele está perto.

* * *

Você deve estar se perguntando neste momento: "Mas o que é especial em tudo isto sobre pequenos projetos? Se não *tudo* equipas de projeto definir essas propriedades no lugar?"

A resposta - com duas notas laterais - é: "Claro." As propriedades que compõem um projeto de pequena equipas bem sucedida *devemos* ser muito semelhante a fazer qualquer projeto bem sucedido, mas otimizado para a situação de pequena projeto.

A primeira nota é que as propriedades são mais fáceis de alcançar em um projeto pequeno: Segurança pessoal é mais fácil, já que as pessoas interagem umas com as outras com mais frequência e vir a conhecer uns aos outros antes. Os laços de realimentação são muito menor, e o resto das propriedades seguir em conformidade.

A segunda nota é que Osmótica Comunicação, que vive da audiência fundo e comunicação ao longo de linhas de visão, realmente só funciona com pequenas equipas. Maior equipa irá configurar Comunicação osmótica dentro subequipas e Fechar Comunicação através subequipas.

Capítulo 3

Na prática (Estratégias e Técnicas)

O Crystal Clear não requer qualquer estratégia ou técnica. É bom ter um conjunto na mão para começar, no entanto. Este capítulo apresenta alguns do menos bem documentado e mais significativa usada por equipas modernas de desenvolvimento ágil.

Muitas estratégias e técnicas úteis foram nomeados na última década, de "jogo de planificação" do XP para Dave Thomas e Andy Hunt "balas traçantes." Os livros por Dave Thomas e Andy Hunt (*Programador pragmática),* Kent Beck (*Programação Extrema Explicada),* e Martin Fowler (*refactoring)* são particularmente rico em ideias.

Existem algumas estratégias e técnicas não apontaram naqueles livros que são úteis para a equipas do projeto Crystal Clear, particularmente em guiar seu caminho através dos primeiros meses do projeto e chegar ao primeiro dos seus Entregas frequentes. I incluem alguns aqui que fazer um bom conjunto, são recomendadas pelos desenvolvedores experientes, poucas pessoas parecem saber, são simples, e acima de tudo, são úteis.

as estratégias

As estratégias que selecionei para contorno são:

1. ***Exploratórios 360 °,*** **parte de fretamento projeto,**
2. ***No início da vitória,*** **uma estratégia de gestão de projetos,**
3. ***caminhando de esqueleto*** **e**
4. ***Incremental rearquitetura,*** **estratégias relacionadas para priorizar o trabalho nas iterações iniciais,**
5. ***Radiadores de informação,*** **uma estratégia para a comunicação.**

Estratégia 1. **Exploratórios 360 °**

No início de um novo projeto, geralmente durante o *fretamento* atividade, a equipas precisa estabelecer que o projeto é tanto significativo e eles podem entregá-lo usando a tecnologia destina. Eles olham ao seu redor em todas as direções, a amostragem do projeto

- **valor do negócio,**
- **requisitos,**
- **modelo de domínio,**
- **planos de tecnologia,**
- **plano de projeto,**
- **a composição da equipas,**
- **processo ou metodologia (ou convenções de trabalho). (Eles podem verificar outros aspectos do projeto, mas estes são os de costume.) O todo *Exploratórios 360 °* para um projeto de Crystal Clear leva alguns dias até uma semana ou duas, se alguma tecnologia nova e peculiar é para ser usado. Com base no que eles aprendem, eles decidir se faz sentido continuar ou não.**

* * *

Valor do negócio **amostragem consiste de captura, com os principais interessados, o que o sistema deve fazer para seus utilizadores e sua organização (s). Isso deve resultar nos nomes dos casos de uso importantes para o sistema, juntamente com os papéis focais o sistema deve servir, as personalidades e funções que devem apresentar ao mundo.**

requisitos **amostragem consiste em casos de uso de baixa precisão que mostram o que o sistema deve fazer, e com o que as outras pessoas e sistemas que terá de interagir. Muitas vezes que o exercício elaboração transforma-se interfaces entre organizações ou sistemas de tecnologia que anteriormente não tinham sido identificados.**

Concorrentemente ou a partir dos esboços de casos de uso, os desenvolvedores provar a *domain-modelo.* Esta amostra serve para destacar os conceitos-chave que irão trabalhar com, o núcleo do negócio, a programação e vocabulário discursiva. Ele também ajuda a equipas para estimar o tamanho e a dificuldade do problema na mão.

Os desenvolvedores criar uma *tecnologia* amostragem, executando alguns experimentos com a tecnologia. Ward Cunningham e Kent Beck chamar estes *Espigões (* veja http://c2.com/cgi/wiki?SpikeSolution). A atribuição é perguntar: Será que podemos realmente conectar essas tecnologias? Será que eles vão suportar as cargas destinadas? Pode nossos desenvolvedores dominar as tecnologias em tempo? Os programadores executar a atribuição de programação menor necessária para deixar claro que o projeto não está correndo por um beco sem saída. Os experimentos estabelecer a plausibilidade técnica do projeto.

A equipas cria uma granulação grossa *plano de projeto,* possivelmente a partir do mapa de projeto ou um conjunto de histórias e lançamentos. Ele pode ser feito usando a técnica de blitz de planificação descrito posteriormente neste capítulo, ou jogo de planificação do XP. Este plano é revisto pela *Designer-chefe,*

a *Patrocinador executivo* e a *Embaixador Utilizador* para garantir que o projeto está agregando valor ao negócio apropriado para despesa adequado em um período de tempo adequado.

Finalmente, os desenvolvedores discutem seu processo, em um *workshop de reflexão* ou um *Miniature processo.*

*** * ***

Aqui está a história de um projeto que falhou três elementos do *Exploratórios 360 °:*

> **Durante a amostra de tecnologia, os desenvolvedores descobriram, para sua surpresa, que não podiam conectar o sistema de e-mail da organização para sua intranet e sistema de browser. Não ficou claro como eles poderiam realmente entregar os serviços de software que tinham imaginado.**
>
> **A amostra de planificação do projeto mostrou o custo ser cerca de três vezes maior do que o *Patrocinador executivo* estava interessado em gastos.**
>
> **A amostra valor do negócio mostrou que a organização não deve alocar muitos recursos para desenvolvedores para este problema; Seria melhor terceirizada (ou simplesmente comprado), e a equipas de desenvolvimento atribuído a um problema de negócios mais significativo.**

Você poderia pensar que o cancelamento ou a terceirização de um projeto como este seria óbvio para todos os envolvidos. No entanto, os desenvolvedores estavam ansiosos para experimentar a nova tecnologia e, portanto, manteve o projeto vivo, sob o pretexto de que este projeto serviria como um bom veículo de aprendizagem. Felizmente, os patrocinadores executivos deram atenção à *Exploratórios 360 °* resultados, parou os programadores, terceirizou o projeto e colocar esses desenvolvedores-chave em um projeto de muito maior valor para a organização (que, estou feliz para adicionar, eles gostaram muito mais).

Estratégia 2. **Victory início**

Ganhando **é uma força que une uma equipas e contribui para a auto-confiança de seus utilizadores. Sociólogo Karl Weick (1977 ???) constatou que *pequenas vitórias* ajuda um grupo a desenvolver força e confiança. Felizmente, estes podem ser organizadas relativamente no início do projeto, quando a equipas mal precisa deles. Isto é o *Victory início* estratégia.**

Em projetos de software, o *Victory início* a procurar é a primeira peça de visivelmente a execução de código, testado. Este é geralmente o *Caminhando de esqueleto (* um pequeno pedaço de função do sistema utilizável, muitas vezes não muito mais do que a capacidade de adicionar um item para o banco de dados do sistema e, em seguida, olha para ele). Embora isto possa não parecer muito, membros da equipas de aprender com esta pequena vitória estilos de trabalho uns dos outros, os utilizadores obter uma visão precoce do sistema, e os patrocinadores ver a equipas entregar.

* * *

As equipas de projeto muitas vezes discutem sobre a seqüência em que para atacar o problema. Uma estratégia frequentemente mencionado é *Pior coisa em primeiro lugar.* O raciocínio é que uma vez que a pior coisa é fora do caminho, tudo será mais fácil.

O problema com a pior coisa-primeiro é que se a equipas não entregar, o patrocinador não tem idéia de aonde a falha reside: É a equipas não é bom o suficiente para retirar este projeto, é a tecnologia de errado, ou é o processo de errado? Além disso, os membros da equipas podem ficar deprimido ou começar a discutir uns com os outros.

Desde que muitas vezes se juntar equipas que não tenham trabalhado juntos antes e estão enfrentando um novo problema com a nova tecnologia, eu prefiro *Coisa mais fácil Primeiro, Hardest segundo lugar.* Os membros da equipas começa a depurar

sua comunicação e seu processo em uma atribuição relativamente simples. Eles e os patrocinadores obter a confiança de um *Victory cedo.* Se o problema mais difícil ainda está fora capacidades da equipas, eu olho para o *coisa mais difícil que a equipas pode ter sucesso com* como a segunda tarefa.

Uma vez que o risco de equipas e falha técnica diminui, uma boa estratégia é *Maior Valor Business First.* Esta estratégia não só gera retornos financeiros máximos para o projeto (SBN 2004 ???), mas bem mapeado para gráficos de valor ganho 18. Você pode *mostrar* todos que você está agregando valor ao negócio e não apenas trabalhar horas. Se o

projeto deve ser executado fora do tempo, será claro para os patrocinadores que eles têm o melhor valor para o dinheiro no tempo gasto, útil em ambas as circunstâncias amigáveis e hostis19.

18 Veja p. 118.

19 Veja candidaturas de projetos de preço fixo mal formadas, na p. 309, e também (Patton 2003).

Estratégia 3. **caminhando de esqueleto**

UMA *caminhando de esqueleto* é uma pequena aplicação do sistema que executa uma função pequena extremidade-a-extremidade. Ele não precisa usar a arquitetura final, mas deve unir os principais componentes da arquitetura. A arquitetura e a funcionalidade pode então evoluir em paralelo.

* * *

Eu aprendi primeiramente do *caminhando de esqueleto* ideia quando um projeto de *Designer-chefe* se aproximou de mim para uma conversa e descreveu um projeto dele. Ele disse (aproximadamente):

> **Tivemos um grande projeto para fazer, consistindo de sistemas que passam mensagens em torno de um anel para outro. O outro líder técnico e eu decidimos que deveríamos, na primeira semana, conectar os sistemas em conjunto para que eles pudessem passar uma única mensagem nula em torno do anel. Desta forma, pelo menos teve o funcionamento do anel.**
>
> **Nós, então, necessário que, no final de cada semana, não importa o que as novas mensagens e mensagem de processamento foi desenvolvido durante a semana, o anel tinha que estar intacto, passando todas as mensagens das semanas anteriores sem falhas. Desta forma, poderíamos crescer o sistema de uma forma controlada e manter as equipas diferentes em sincronia.**
>
> **Funcionou maravilhosamente, e gostaríamos de fazê-lo novamente.**

Eu tive a oportunidade de aplicar essa idéia em um sistema cliente-servidor no meu primeiro grande projeto.

> **Nós estávamos movendo assustadoramente lentamente. Nossa primeira entrega de três meses foi programado para entregar apenas um pequeno conjunto de funcionalidades que pode ser considerado interessante para os utilizadores finais. Tivemos que cortar escopo para atender até mesmo essa ambição leve, e, no final, entregue o que eu chamo de "ler um número, escreva um número."**
>
> **Para minha surpresa, o sistema entrou no ar na programação, os patrocinadores jogou uma grande festa, e os utilizadores começaram a colocar os números no sistema.**
>
> **desenho e código do sistema não era bonita, mas tinha conectado o software do lado do servidor para o sistema de banco de dados e poderia escrever funções em cima do que arquitetura. Tendo os elementos de arquitectura ligados e uma peça de amostra de função de execução em que, fomos capazes de desenvolver mais funcionalidade em paralelo com a revisão da arquitectura para ser mais robusta (o *rearquitetura incremental***
>
> **estratégia).**
>
> **Nossos patrocinadores foram satisfeito com este *Victory cedo.* Testamos a equipas, o processo, as tecnologias, e a arquitetura em um ponto muito no início do projeto.**

Outros autores têm outros nomes para tipos semelhantes de ideias. Os Poppendiecks falam sobre uma "aplicação que mede" (Poppendieck 2003), Dave Thomas e Andy Hunt usar o que eles chamam de "balas traçantes" (Caça 2001 ???).

O que constitui um *caminhando de esqueleto* varia com o sistema que está sendo projetado. Para um sistema cliente-servidor, seria um único capacidade de tela de-to-database-e-volta. Para um multi-tier ou sistema multi-plataforma,

é uma conexão de trabalho entre os níveis ou plataformas. Para um compilador, que consiste de compilação do elemento mais simples da linguagem, possivelmente apenas um único token. Para um processo de negócio, ele está andando através de um único e simples transação comercial (como Jeff Patton descreve na técnica

Interaction Design Essencial).

Note-se que cada subsistema é incompleta, mas eles estão ligados entre si, e vai ficar viciado em conjunto a partir deste ponto.

o *caminhando de esqueleto* não está completa ou robusta (só *anda em,* perdoem a expressão), e está faltando a carne do funcionalidade do aplicativo. Incrementalmente, ao longo do tempo, a infra-estrutura será concluído e será adicionado a funcionalidade completa.

UMA *caminhando de esqueleto* é diferente de um *Espinho 20.* Um aumento é "a mais pequena aplicação que demonstra sucesso técnico plausível." O pico normalmente demora entre algumas horas e alguns dias para ser concluído, e é jogado fora depois, desde que foi construído com hábitos de codificação de não-produção. Um aumento serve para responder à pergunta: "Será que estamos indo na *errado* direção?"

UMA *Caminhando de esqueleto,* por outro lado, é um código permanente, construído com hábitos de codificação de produção, testes de regressão, e destina-se a crescer com o sistema. Uma vez que o sistema está instalado e funcionando, ele vai ficar instalado e funcionando para o restante do projeto, apesar da *rearquitetura incremental* que é bastante provável de ocorrer.

20 htt:? //c2.com/cgi/wiki SpikeSolution

Estratégia 4. **rearquitetura incremental**

A arquitetura do sistema terá de evoluir, a partir do *Caminhando de esqueleto,* e também para lidar com tecnologia e negócios mudanças de requisitos ao longo do tempo. Raramente é eficaz para desligar desenvolvimento para realizar uma revisão de arquitetura, então a equipas aprimora a arquitetura em etapas, mantendo o sistema funcionando como eles fazê-lo.

A equipas aplica a idéia de desenvolvimento incremental para a revisão da infra-estrutura ou arquitetura, bem como a funcionalidade final do sistema.

*** * ***

A pergunta surge naturalmente: Como totalmente projetado deve a arquitetura e infra-estrutura do sistema ser durante as fases iniciais do projeto?

Qualquer pessoa tem o seu pessoal "horizonte pensei," quanto a complexidade que ele pode manter em sua cabeça por vários dias em uma fileira, o quanto ele pode prever a partir de sua experiência e conhecimento da situação. Um arquiteto que fez sistemas semelhantes no mesmo domínio usando tecnologias similares podem pensar através de um novo horizonte de um apenas começando.

Algumas pessoas mantêm o horizonte pensamento para baixo para alguns dias. Eles começar imediatamente com um projeto inicial, e aprender com essa versão cedo em qual direção eles devem empurrar o design. Outros gostam de pensar mais e considerar mais contingências antes de cometer a uma arquitetura inicial.

O horizonte de pensamento em um projeto de Crystal Clear é quase certamente alcançado dentro de uma semana ou duas. Nesse ponto, os designers estão provavelmente especular além do seu horizonte de pensamento, e seria melhor configurar o *Caminhando de esqueleto.* Eles devem usar isso para chegar à primeira de seu As entregas frequentes, e usar o feedback para melhorar a arquitetura. "Não ultrapassagem seus faróis," é uma frase pessoas ele.

Aqui está uma história sobre o uso de *rearquitetura incremental* do meu primeiro grande projeto.

> **A equipas de infraestrutura encontrado após a segunda entrega que o mapeamento de banco de dados objeto-torelational eles tinham planejado usar exigiria uma quantidade sempre crescente de programação como nova funcionalidade foi adicionada; em outras palavras, ele não escala. Estar sob pressão de tempo pesado em um contrato de preço fixo, a pressão estava sobre eles simplesmente manter arar a frente e apenas trabalhar mais e mais rápido para manter-se com o aumento da carga de trabalho.**
>
> **O líder da equipas decidiu, no entanto, que uma reformulação arquitetônica foi necessário. Sua equipas continuou a apoiar o design antigo por mais um incremento, enquanto a funcionalidade entregue do sistema ainda era pequeno, e ao mesmo tempo começou a sua reformulação. Eles deslizou a nova arquitetura para o terceiro ciclo de entrega, treinando as equipas função de desenvolvimento sobre como escrever para as novas interfaces, e apoiá-los com algum número de mecanismos automáticos de geração de código. no quarto ciclo de entrega, eles arrancado todos os usos da arquitetura inicial.**
>
> **No quarto ciclo de entrega, o arquiteto de infra-estrutura deu um suspiro de alívio: eles poderiam finalmente manter-se com a funcionalidade do sistema agora rapidamente.**

O inverso também se aplica. Aqui está a história infeliz de uma equipas que não se aplicava *rearquitetura incremental* quando eles deveriam ter.

> **Os arquitetos prometeu seus executivos de que sua arquitetura radicalmente nova permitiria tradução direta casos de uso para a execução de código Java, tornando o produto extremamente sensível às mudanças nas necessidades de negócios. Esta abordagem foi, naturalmente, uma proposta arriscada, já que ninguém tinha feito isso antes.**
>
> **Reconhecendo esse risco, eu sugeri ao gerente de projeto que seu projeto usar a estratégia exata que tinha usado na história um pouco acima: criar uma arquitetura simples e direta que pode ser entregue no prazo, e swap na nova arquitectura se e quando ele provou em si.**
>
> **O gerente do projeto disse que ela tinha grande confiança nos arquitetos de chumbo e não queria que o retrabalho extra, então ela escolheu para pendurar todas as esperanças na nova arquitetura. Infelizmente, o problema de ir de casos de uso de código de execução acabou por ser insolúvel a essas pessoas. No final, o projeto foi deixado com nenhuma arquitetura em execução em tudo, e nenhum produto já foi enviado.**

Desenvolvedores na última década têm mostrado que arrumado arquiteturas, simples são relativamente simples de atualizar para o próximo estágio de complexidade e desempenho. A conseqüência negócios desta situação é que, muitas vezes, uma empresa pode criar uma versão inicial para demonstrar a função e possivelmente até mesmo gerar receita, e, em seguida, usar o *rearquitetura incremental* estratégia para renovar a infra-estrutura sob o sistema em funcionamento.

A partir de uma arquitetura de trabalho simples e aplicando *rearquitetura incremental* é uma estratégia vencedora para a maioria, embora nem todos os sistemas 21. Ele fornece uma série de vantagens quando ele pode ser usado:

- **A arquitetura é mais fácil modificar quando são necessárias modificações.**
- **Os Função- e infra-estrutura de equipas começar a trabalhar em paralelo, avançando o momento em que a função útil se torna disponível, ou pelo menos visível.**
- **Os utilizadores finais podem visualizar a funcionalidade proposta do sistema de madrugada, e corrigi-lo é a aptidão fundamental para o uso do negócio.**
- **O sistema em execução pode revelar deficiências na arquitetura que os experimentos mentais primitivos não pegar.**
- **O sistema pode, por vezes, ser implantado em um mercado limitado, caso em que a empresa vai começar a ganhar receitas para ajudar a pagar para o desenvolvimento contínuo. Ao decidir quando aplicar a estratégia, pergunte**
- **se o sistema pode, de fato, ser desenvolvido em duas correntes, com a infra-estrutura ou arquitetura evolui em paralelo com a funcionalidade do sistema (a resposta é geralmente Sim);**

---

21 Seria, por exemplo, provavelmente seria uma má idéia para pedir a todos os utilizadores de telefone celular para tirar seus telefones de volta para o fornecedor para que uma nova arquitetura de software interno poderia ser baixado!

- **se, primeiro criando um fallback, arquitetura simples e testar o sistema em uso ao vivo, o negócio pode pegar um erro arquitetônico imprevisíveis (a resposta é muitas vezes sim);**

- **se o negócio pode gerar receita no início implantação de uma versão com funcionalidade limitada a um mercado limitado (a resposta é surpreendentemente, muitas vezes Sim).**

rearquitetura incremental **é discutido longamente no artigo "Estendendo um Arquitetura como ele ganha valor comercial" (Cockburn 2004).**

Estratégia 5. **Radiadores de informação**

A *informações Radiator* é uma exposição postado em um lugar aonde as pessoas podem vê-lo como eles trabalham ou passam. Ele mostra os leitores informações que eles se preocupam sem ter que pedir a alguém uma pergunta. Isso significa mais comunicação com menos interrupções.

Uma boa informação radiador

- **é grande e facilmente visível para o observador casual, interessado,**
- **é entendido de relance,**
- **mudanças periodicamente, de modo que vale a pena visitar,**
- **é facilmente mantido até à data.**

Normalmente, é em papel, afixado na sala de equipas ou no corredor. Em algumas circunstâncias excepcionais, é em uma página da web que as pessoas referem-se com frequência. exemplos incomuns de radiadores de informação incluem uma (real!) semáforo, uma esfera colorida, e um monitor de computador pendurado do lado de fora da partição de um cubículo para o corredor.

Todd Pouco da Landmark Graphics fez a observação interessante que radiadores de informação geralmente servem para informar as pessoas *lado de fora* a equipas do projeto. As pessoas da equipas do projeto geralmente sabem as informações postadas, por causa de suas comunicações estreitas entre si. É as pessoas fora da equipas que querem ou precisam de saber essas informações a fim de tomar suas próprias decisões, e de outra forma iria interromper a equipas para obter essa informação, ou simplesmente adivinhar o que (muitas vezes incorretamente).

* * *

Radiadores de informação **pode ser usado em qualquer projeto, grande ou pequeno. Uma pequena equipas pode usá-los convenientemente para manter informações que de outra forma teriam de manter no computador (que é tanto mais lento e menos visível).**

radiadores de informação normalmente são usados para mostrar informações de status, como

- **conjunto de trabalho da iteração atual (casos de uso ou histórias),**
- **as atribuições de trabalho atual,**
- **o número de testes de escrita (ou transmitido),**
- **o número de casos de uso (ou histórias) entregue,**
- **o status de servidores de chaves (para cima, para baixo, na manutenção),**
- **o núcleo do modelo de domínio,**
- **os resultados da última *Workshop de reflexão.***

arquivos online e página web geralmente não fazem bons radiadores de informação, porque um radiador de informações precisa ser visível sem esforço significativo por parte do espectador. Eu, é claro, correr em exceções a esta regra. Uma delas foi a equipas executora *Integração contínua* com CruiseControl, um servidor dedicado executandoum script build-e-teste e postar os resultados do teste para uma página web. Os programadores tendem a deixar essa página web visível em todos os momentos em suas telas, para que eles**

poderia responder a falhar os testes de integração imediatamente. A outra excepção foi o monitor pendurados sobre uma parede do compartimento para o corredor (ver Figura 3-5), que indica as medições do tempo de execução de corrente do sistema, em utilização.

Freeman-Benson e Borning escreveu um relato de experiência sobre uma metodologia que eles chamam YP (Freeman-Benson 03). Eles relatam sobre o efeito do uso de um semáforo real:

> **Na verdade, existem várias: um no corredor do nosso laboratório, dois a mais em escritórios de desenvolvedores, e um semáforo virtual na web. Nós usamos quatro combinações de cores:**
>
> - **verde quando a compilação e todos os testes foram bem sucedidos,**
> - **amarelo quando a compilação e testes estão em andamento,**
> - **amarelo e verde em conjunto quando a compilação passou o ponto em que é susceptível de falhar (na prática, isto significa que todos os testes tenham passado e que o instalador e distribuição final estão a ser produzido), e**
> - **vermelho se qualquer parte da compilação ou testes falhou.**
>
> **O semáforo é um símbolo poderoso do estado atual do software. A versão web em www.urbansim.org/fireman torna o estado visível para qualquer outra pessoa que possa estar interessado (incluindo você, caro leitor, se quiser), e em particular para os nossos clientes, embora de uma forma menos convincente do que o físico dispositivo.**
>
> **O primeiro autor usou o semáforo como parte da aplicação YP em quatro outros projetos de desenvolvimento também. Curiosamente, apenas uma equipas, além de nosso ainda está usando a luz - as outras equipas desligado as luzes porque não gosta deles sendo vermelho o tempo todo. Ao invés de corrigir o problema subjacente (que seu sistema era suficientemente instável que seus testes de regressão não passaria de forma confiável), eles optaram por "eliminar o mensageiro." Apenas uma das equipas reconheceu esta decisão explicitamente.**
>
> **Em conversas com outros líderes de equipas e gerentes de software, aprendemos que um número deles tinha tentado "indicadores de falha", como sirenes, luzes piscando, luzes vermelhas, e assim por diante, e que cada um tinha cancelado o experimento depois de alguns dias. Aparentemente, a utilização de reforço de negativo (uma luz vermelha) sem o reforço positivo correspondente (uma luz verde) era demasiado prejudicial para o moral. A nossa experiência com o semáforo é completamente o oposto: todos que se junta uma equipas YP reporta imediatamente uma sensação de conforto na grande (8-12") luz verde brilhando sua mensagem de 'tudo está bem com a compilação' Ou na. raras ocasiões em que o software falhou os testes de build noturnos, como o pessoal chega na parte da manhã, na escuridão do inverno, com o corredor laboratório iluminado pelo brilho vermelho do semáforo,**

Abaixo estão fotografias de diferentes radiadores de informação.

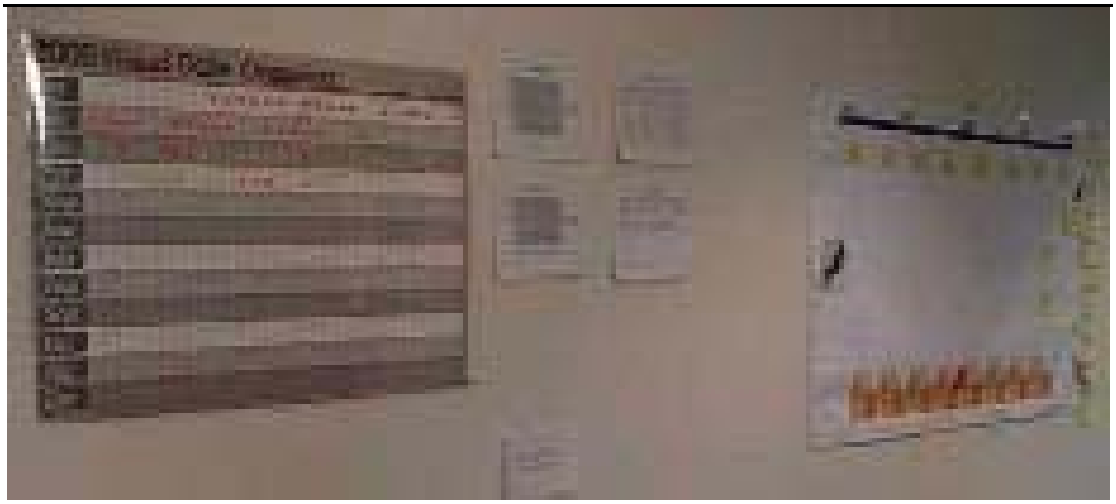


Figura 3-1. estado de desenvolvimento das histórias de utilizadores. (Graças a ThoughWorks)

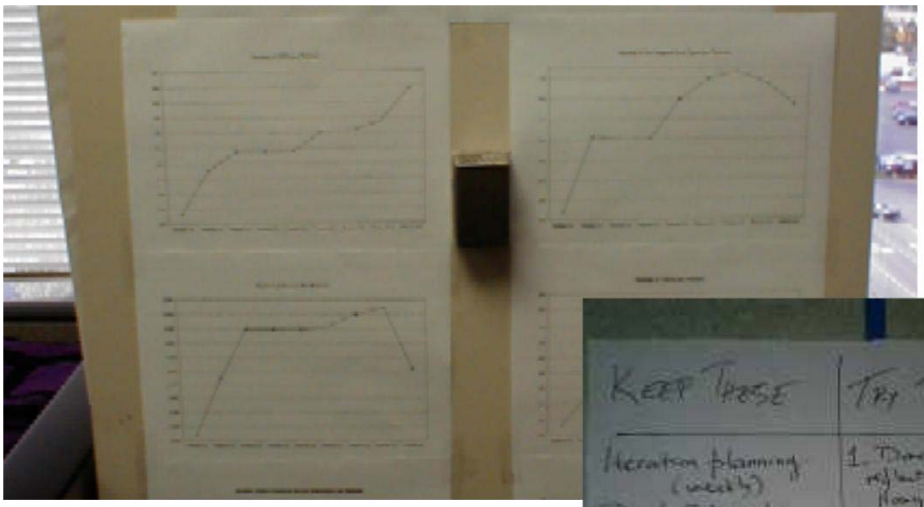


Figura 3-2. Gráficos da base de código mantida perto dos programadores. Observe o tamanho do código que vai para baixo nas duas últimas entregas! (Graças a Randy Stafford)

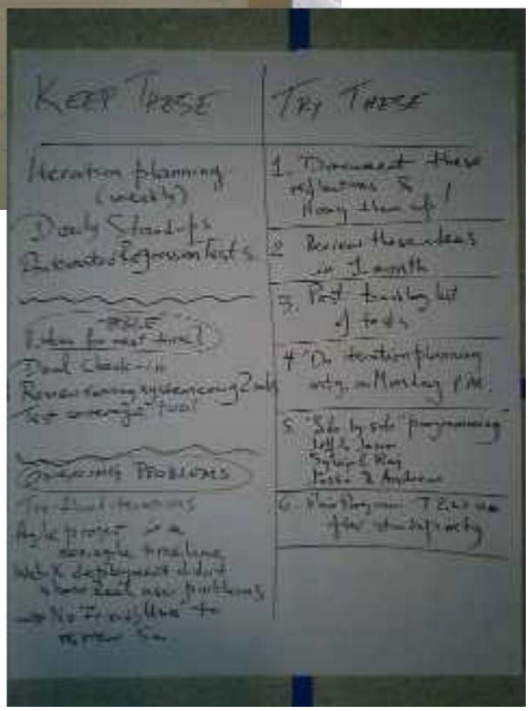


Figura 3-3. Os resultados de uma oficina de reflexão. (Graças a Jeff Patton)

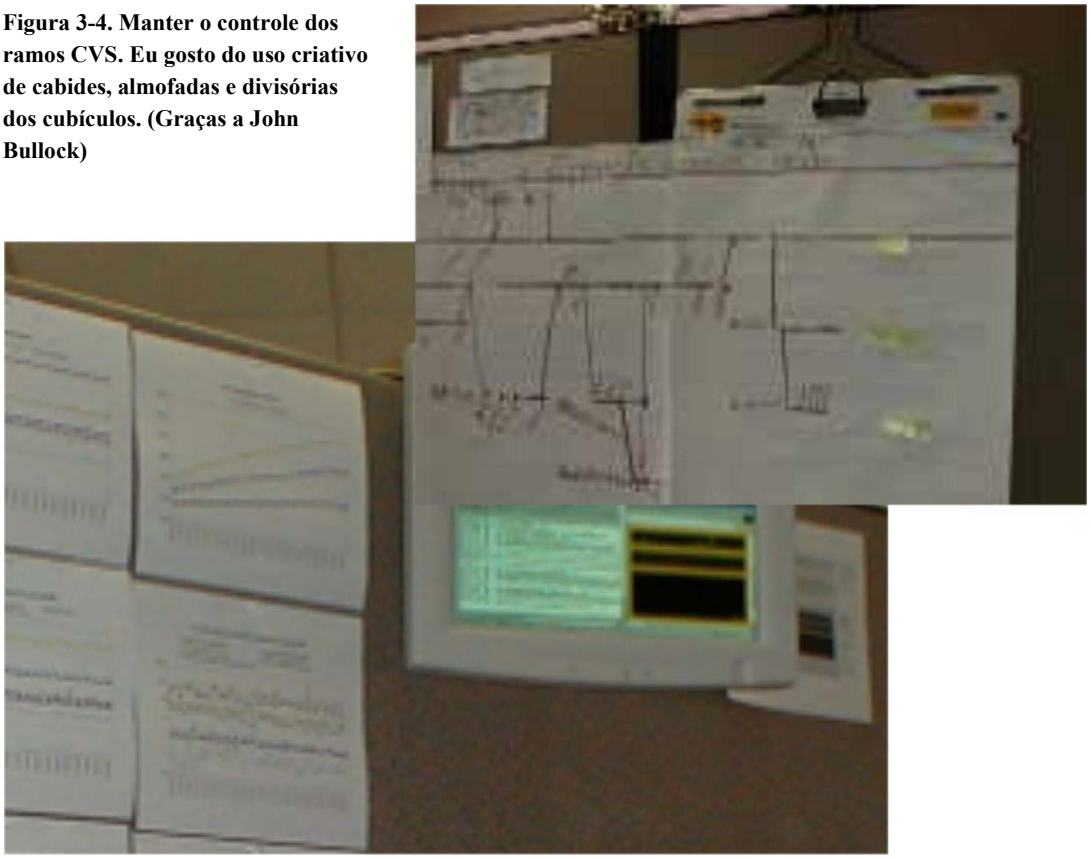


Figura 3-4. Manter o controle dos ramos CVS. Eu gosto do uso criativo de cabides, almofadas e divisórias dos cubículos. (Graças a John Bullock)

Figura 3-5. Um monitor pendurado sobre a parede do cubículo para mostrar medições em tempo real do sistema em operação. (Graças a Randy Stafford)

as Técnicas

As técnicas que são selecionados

1. ***Metodologia Shaping,*** **coleta de informações sobre experiências anteriores e usá-lo para vir acima com as convenções de partida,**
2. ***Workshop de reflexão,*** **um formato de workshop especial para Melhoria reflexivo,**
3. ***Blitz Planificação,*** **que também às vezes se referem como a elaboração de projetos "jam session" (Como no jazz) para enfatizar a sua natureza colaborativa, uma técnica de planificação de projeto rápida e colaborando,**
4. ***Delphi Estimativa,*** **uma maneira de chegar a uma estimativa inicial para o total do projeto,**
5. ***Stand-ups diários,*** **uma maneira rápida e eficiente para passar informações ao redor da equipa em uma base diária,**
6. ***Interaction Design ágil,*** **uma versão rápida de design centrado no uso,**
7. ***Miniature processo,*** **uma técnica de aprendizagem,**
8. ***Side-by-Side de programação,*** **uma alternativa menos intenso para emparelhar programação,**
9. ***Queime gráficos,*** **uma maneira eficiente de planejar e relatar o progresso, particularmente adequado para uso em *Radiadores de informação.***

Conforme descrito na série de e-mail entre Cristal e Alistair, nenhum específico, a técnica é obrigatória por Cristal. Estas estratégias particulares oferecem um bom ponto de partida, particularmente a formação metodologia e workshops de reflexão, uma vez que estes são mais susceptíveis de ser novo para o grupo. Queimam gráficos tornaram-se um tema interessante em seu próprio direito, uma vez que têm uma partida natural para as cartas de valor ganho utilizados na gestão de projetos de engenharia de sistemas.

Técnica 1. **metodologia Shaping**

Estabelecer a metodologia de partida em duas etapas:

1. entrevistas do projeto

2. Metodologia moldar oficina As informações do primeiro alimenta o segundo.

Entrevistas projeto

Com esta técnica pouco, você vai construir uma pequena biblioteca de experiências na sua organização que mostram as forças, fraquezas e temas de sua organização. Quando você vai para a oficina de metodologia de formação, você vai examiná-los e discutir como tirar proveito dos pontos fortes e como compensar, ou evitar, as fraquezas.

> **Encontramos em uma organização um tema que projetos que tiveram uma boa comunicação interna e externa foi bem, mas quando eles não têm essa comunicação, as equipas tiveram um mau tempo e o resultado foi negativo.**
>
> **Vendo isso na frente de nós, eu fiz um esforço extra para trazer nossas experiências do projeto para os diretores de desenvolvimento e da divisão requerente. Nós escreveu um resumo de uma página após cada iteração, resumiu nossos custos, realizações, frustrações e lições aprendidas. Nos organizamos para essas duas pessoas para atender por uma hora e rever o projeto usando o relatório como base para a sua discussão. Eles realmente falou sobre muitas outras coisas durante essa hora, mas ambos comentou sobre os efeitos positivos da alocação de tempo para discussão, e começando passando pelo relatório.**
>
> **No final daquele ano, o diretor de desenvolvimento comentou, um pouco intrigado, que tínhamos tido uma atribuição projeto bastante semelhante como dois outros grupos, mas os outros dois projetos falharam, enquanto o nosso sucesso. Olhando para os projetos, percebemos que o nosso era o único projeto que levou tempo para atender a caminhos de comunicação dentro e externos ao projeto. O tema tínhamos encontrado em nossas entrevistas projeto tinha jogado para fora mais uma vez, fiel à forma.**

Técnica Variante 1

Comece com si mesmos, mas também incluem outras pessoas de alguns outros projetos em sua organização. Tem várias pessoas em sua equipas entrevistar pessoas em outros projetos. Comece sua coleção com quatro a dez relatórios de entrevista. É útil (mas não crítica) que cada um de seu povo entrevistar mais de uma pessoa em um projeto. Você pode falar com qualquer um dos dois o seguinte: o gerente do projeto, o líder da equipas, um designer de interface do utilizador, e um programador. Suas diferentes perspectivas sobre o mesmo projeto irá provar informativo. Ainda mais informativo, no entanto, serão as respostas comuns em vários projetos.

Tenha em mente é que tudo o que o entrevistado diz é relevante. Durante uma entrevista, não oferecem suas próprias opiniões sobre qualquer assunto, mas usar o seu julgamento para selecionar uma próxima pergunta a fazer.

As etapas a seguir resumem como eu tenho trabalhado ao longo dos anos em fazer minhas próprias entrevistas do projeto (Cockburn, 2003).

Passo 1. **I pedir para ver uma amostra de cada produto do trabalho produzido. Olhando para estes, eu detectar o quanto cerimônia era susceptível de ser sobre o projeto e pensar sobre o que perguntas devo perguntar sobre os produtos de trabalho. Eu olho para o trabalho duplicado, lugares aonde eles poderiam ter sido difícil manter-se atualizado. Pergunto se o desenvolvimento iterativo estava em uso, e em caso afirmativo, como os documentos foram atualizados no seguinte iterações. Eu olho, em particular, de maneiras em que a comunicação informal foi usado para corrigir sobre inconsistências na papelada.**

> **Em um projeto, o líder da equipas me mostrou 23 produtos de trabalho. Notei um bom grau de sobreposição entre eles, e então eu perguntei se os posteriores foram gerados pelas ferramentas de que os anteriores. O líder da equipas disse que não, as pessoas tinham para reinserir-los do zero. Então, eu seguido por perguntando como as pessoas se sentiam sobre isso. Ele disse que eles realmente odiava isso, mas ele fez-lhes fazê-lo de qualquer maneira.**

Você pode adivinhar que esse último pedaço de informação é muito útil na oficina metodologia de formação.

Passo 2. **Peço uma curta história do projeto. Esta história inclui data de início, mudanças de pessoal (crescimento e encolhimento), estrutura da equipas, os pontos de vida do projeto emocionalmente altas e baixas. Eu faço isso para calibrar o tamanho e tipo de projeto e para detectar aonde pode haver interessantes outras perguntas a fazer.**

Etapa 3. **Eu pergunto, "** ***Quais foram as principais coisas que você fez de errado que você não gostaria de repetir em seu próximo projeto?"*** **Eu escrevo o que eles dizem, e eu pescar em torno de questões relacionadas para investigar.**

Passo 4. **Eu pergunto, "** ***Quais foram as principais coisas que você fez certo que você certamente gostaria de preservar em seu próximo projeto?"*** **Em resposta a esta pergunta, as pessoas têm chamado de tudo, desde aonde se sentam, para ter comida na geladeira, para atividades sociais, canais de comunicação, ferramentas de software, arquitetura de software, e modelagem de domínio. Tudo o que você ouvir, anotá-la.**

Passo 5. **I revisitar as questões "** ***Quais são as suas prioridades em relação às coisas que você gostou no projeto? O que é mais crítica para manter, eo que é mais negociável?"*** **Isso é redundante, tecnicamente falando, mas minha experiência é que as pessoas vêm com um pouco diferentes respostas. Eu escrevo os para baixo. É útil perguntar neste momento, "** ***Houve alguma coisa que te surpreendeu sobre o projeto?"***

Passo 6. **Finalmente, pergunto se há alguma coisa que eu deveria ouvir falar. Eu vejo aonde pergunta vai.**

Você pode achar que é útil para construir um modelo de entrevista em que para escrever os resultados, assim você pode trocá-los facilmente. O tempo que nós fizemos isso, nosso modelo foi 2

páginas (para controlar a quantidade de escrever o entrevistador faz) e continha as seguintes seções.

1. Nome do projeto, trabalho da pessoa entrevistada
2.dados do Project (iniciar / datas de término, tamanho pessoal, domínio da tecnologia). história 3. Projeto
4. fez de errado / não repetiria
5. Será que a direita / preservaria
6. Prioridades
7. Outros

Técnica Variante 2

Jens Coldewey, na Alemanha, usou uma técnica diferente para obter a informação que ele procurava para sua oficina metodologia de formação. Ele enviou a cada membro do seu próximo grupo um breve questionário, perguntando o que eles tinha gostado sobre seus projetos anteriores e queria manter no novo projeto, e que eles não tinha gostado sobre seus projetos anteriores e gostaria de configurar de forma diferente sobre a novo projeto. Ele tomou essas respostas direto para a oficina de metodologia de formação.

Técnica Variant 3

Seguindo o exemplo Jens', eu também executar um workshop facilitado para chegar com a mesma informação. Você pode fazer este workshop em grupos de qualquer tamanho, a partir de um projeto de Crystal Clear de apenas quatro pessoas, para uma organização de várias dezenas de pessoas. Permitir uma hora para este workshop se você tiver um pequeno grupo, e duas a três horas para um grande grupo.

Divida em grupos de trabalho de três a cinco pessoas cada. Configurar vários cartazes com canetas de diferentes cores em cada grupo de trabalho.

Tem cada Brainstorm grupo de trabalho e listar todas as coisas que pessoalmente experimentaram em projetos no passado e não gostaria de repetir no projeto atual ou próxima. Eles escrevem todos aqueles por um flipchart (ou, mais provavelmente, várias páginas de que flipchart). Time Box atividade para que eles não vão toda a tarde!

Por outro flipchart e com a outra cor da caneta, tê-los refletir e listar todas as coisas que pessoalmente experimentaram em seus próprios projetos no passado (isto é importante, porque senão eles tendem a anotar ideias que já ouviu falar sobre, mas não experiente), e gostaria de ver repetido.

Passe algum tempo a combinação e fusão das ideias comuns em cada lista para fazer uma lista única de não gostaram / não repetir e uma lista separada de itens / repetição gostava.

Dando a cada pessoa um número, digamos, sete, votos de cada lista, pedir-lhes para marcar em cada lista os itens que eles se sentem mais importante. Eles podem empilhar todos os sete votos em um item ou espalhá-los da forma que quiserem.

Contar os votos para cada item e classificar por resultados da votação.

Neste ponto, você tem uma tabela de conteúdo para situações de projeto para "evitar" e os de "manter", priorizados pelo significado pessoal para as pessoas na sala.

Durante este workshop, você não vai resolver nenhum dos problemas mencionados. Esse é o trabalho a fazer durante o workshop metodologia de formação. O que você tem é a informação necessária em que a próxima oficina, que pode ocorrer no mesmo ou em um dia diferente.

Metodologia Oficina Shaping

O workshop metodologia de modelagem não é mais nem menos do que uma versão maior da oficina reflexão periódica descrito a seguir. Na oficina periódica reflexão, a equipas já tem uma lista de regras e convenções que estão usando, e refletir sobre o que para manter ou deixar cair. Esse workshop deve fazer mudanças relativamente pequenas para o conjunto de convenções.

O workshop metodologia de formação, por outro lado, começa com um grupo de pessoas que não fizeram o exercício antes e ainda não sabe o que suas convenções de funcionamento será. Sua produção será, portanto, uma única grande lista de ideias propostas, regras e convenções. O livro *Desenvolvimento Ágil de Software* contém uma lista de amostra de 36 pontos que era o resultado de uma tal investigação.

O workshop começa com uma revisão das regras fixas da organização sobre desenvolvimento de software. Estes são tomadas como regras dadas, a menos que alguém da equipas tem uma idéia sobre como mudar qualquer um deles.

As pessoas passam pela lista de itens "Achou / manter" que veio das entrevistas projeto e ver quantos desses podem facilmente organizar no próximo projeto. Eles mesa os mais difíceis para estudo separado.

Eles percorrer a lista de itens "não gostava / evitar a" e brainstorm maneiras de evitá-los sobre o próximo projeto.

As respostas são escritas para baixo na lista de ideias, regras e convenções.

Eles localizam os temas da organização (como mostra a história no início desta técnica), e discutir como lidar com, compensar, ou tirar proveito desses temas. Eles escrevem suas respostas na lista de ideias, regras e convenções.

Eles percorrer os itens difíceis salvas até agora, discutir e debater quaisquer formas de lidar com eles, dadas todas as ideias que já tenha escrito para baixo. Eles anotar as ideias que eles querem experimentar na lista principal, e escrever para baixo em uma lista separada os itens que eles estão preocupados com, mas não têm ideias para. (Eles olham para esta lista mais tarde no projeto, seja para ver se eles têm novas ideias, ou para discutir com o patrocinador do projeto e seus gestores, caso se deparam com exatamente esses problemas.)

Neste ponto, a lista de ideias, regras e convenções é provavelmente bastante longo. As pessoas passam pela lista novamente, e marcar os que são mais importantes para prestar atenção, e colocado em parênteses os que são interessantes, mas possivelmente marginal. Estes últimos são os que são bom ter, mas o primeiro a ir em caso a equipas fica sobrecarregado com seguinte da lista.

A equipas irá atualizar essas convenções um mês mais tarde durante o primeiro *Workshop de Reflexão (* Eu escrevo o tempo todo "um mês depois", como um lembrete de que ele não deve ficar muito tempo antes da revisão. O tempo exato escolhido é até a equipas para decidir.)

Acredite ou não, estes são agora os detalhes "Metodologia de arranque" para o projeto!

Técnica 2. **workshop de reflexão**

A equipas deve fazer uma pausa de uma hora periodicamente - certamente depois de cada entrega - para refletir sobre suas convenções de trabalho. Na oficina reflexão, os membros da equipas discutir o que está funcionando bem, que precisa ser melhorado, eo que eles vão fazer de forma diferente durante o próximo período.

Eu gosto de fazer um deles a meio da primeira iteração como uma verificação de sanidade: "Trabalhando desta forma, estamos no caminho para completar a nossa missão nesta iteração, ou precisamos para compensar agora, antes que seja tarde demais? " Desde a minha experiência é que a maioria das equipas agendar muito trabalho a fazer na primeira iteração, presto menos atenção às estimativas ruins de quanto pode ser feito na primeira iteração, e olhar para as convenções de trabalho desastrosamente ruins. Reparação de convenções de trabalho ruins é o que eu focar no primeiro workshop.

Pessoas que possuem oficinas de reflexão em uma base regular me dizer de um padrão que mostra-se: o grupo encontra um monte de discutir nas primeiras oficinas, e então, depois de um tempo, eles acham que há pouco mais a dizer a partir do anterior Tempo. Em alguns casos, eles alongar o tempo entre oficinas, mantendo-os apenas depois de cada entrega. Em outros casos, eles usam o tempo para um exame mais detalhado de seus valores de equipas, estilos de trabalho pessoais e similares.

Existem vários formatos para uma oficina de lidar com esses temas. O que se segue é o meu favorito porque é ao mesmo tempo simples e breve (como são todas as minhas técnicas favoritas). Norm Kerth escreveu um livro chamado *Retrospectivas do projeto* que discute muitos outros exercícios que você pode realizar durante um workshop de reflexão.

O Workshop Reflexão Manter / Try

Muito simplesmente, capturar as seguintes três coisas em um flipchart e, em seguida, publicar o flipchart proeminente como um *informações Radiator* para o grupo para ver como eles funcionam.

1. ***O que devemos ter.*** **Estas são as convenções que não querem perder na próxima período de tempo. Os tipos de coisas que são mencionadas muitas vezes incluem: sentados juntos, horas não-interrompível foco, reuniões diárias de stand-up, e testes de regressão automatizados.**

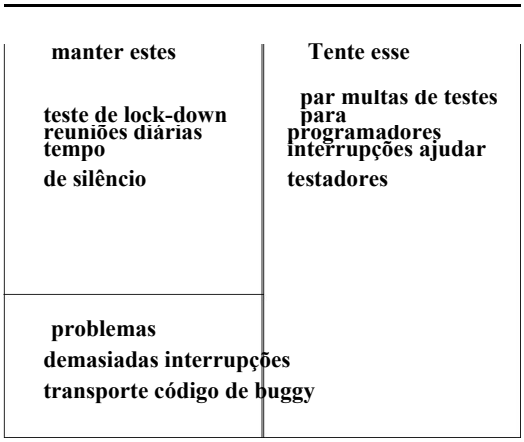


Figura 3-6. formato de cartaz para a oficina de reflexão.

2. ***Aonde nós estamos tendo problemas em curso.*** **Geralmente, não é saudável para se concentrar demais em problemas, mas as pessoas ficam presas na oficina se eles não podem mencionar um problema que está incomodando e vê-lo se escrito no cartaz. Criar uma meia-coluna ou menos no flipchart para capturar problemas que estão tendo problemas para se locomover. Temas que muitas vezes são mencionados são: muitas interrupções, os requisitos de mudança, muitas vezes, as pessoas alterar o código sem notificar ninguém, incapazes de comprar novos conjuntos de ferramentas.**

3.*O que nós queremos tentar no próximo período de tempo.* **Em certo sentido, estes são os mais importante de as três categorias. Estes são o que a equipas concorda em tentar para as próximas semanas. Algumas das sugestões que ouço incluem: duas horas de ininterrupto trabalho de 10 horas até a cada dia; uso de padrões de codificação; mais testes unitários ou de aceitação; ocasional programação em pares; diária reunião stand-up.**

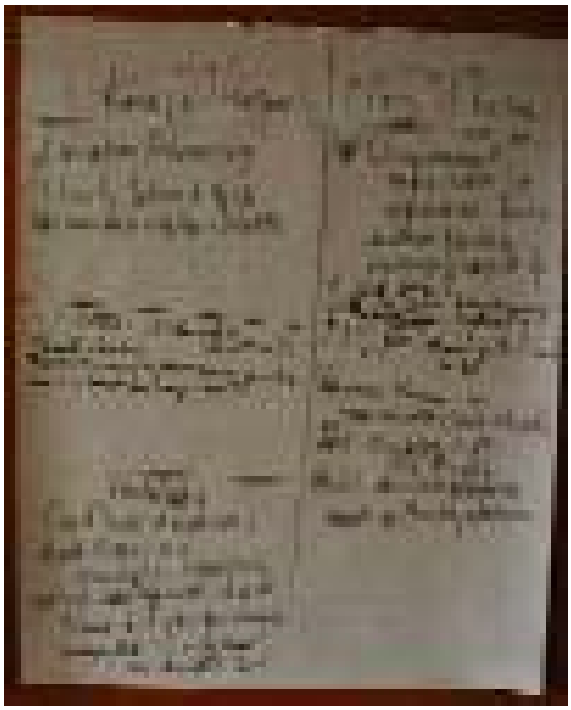
Você pode executar o seu primeiro workshop reflexão em não mais que 15 minutos. Isto irá limitar o tempo da discussão para que as pessoas não correm muito longe, e concentrar a equipa em chegar com sugestões concretas para o que manter eo que mudar. Isso fornece a equipas com um *Miniature processo* para a técnica.

Após a primeira, você pode decidir quanto tempo para alocar para cada workshop. Um grupo escolheu alocar dois dias em um local fora do local a cada três meses para o primeiro ano de seu projeto. Eles usaram o primeiro dia para buildng equipas e oficina de reflexão, e passou o segundo dia planificação do ciclo de entrega próximos três meses. Eles finalmente cortá-la para apenas um dia em seus escritórios. Eu tenho usado uma hora na cafeteria a cada dois meses, até 15 -30 minutos a cada duas semanas.

Aqui estão algumas fotografias de resultados do workshop reflexão usando a técnica acima. Você vai notar no segundo que criamos uma seção adicional pouco acima do *problemas* para capturar ideias que as pessoas tinham, mas não quer tentar de imediato. Esta seção serviu para alimentar a discussão no seguinte workshop.

Figura 3-7. flipcharts oficina reflexão mostrando histórico do projeto e ideias tentou. (Graças a Jeff Patton e Tomax)

Figura 3-8. saída oficina de reflexão. (graças a Tomax e Jeff Patton)



Técnica 3. **Planificação Blitz**

A sessão de planificação é uma oportunidade para o patrocinador executivo, o utilizador embaixador e desenvolvedores de contribuir em conjunto para construir o mapa do projeto e cronograma. Como sempre, há várias maneiras de trabalhar.

Uma técnica é o "jogo de planificação" do XP. No jogo de planificação, as pessoas colocam fichas na mesa, uma história de utilizador por cartão. O grupo finge não existem dependências entre cartões, e simplesmente linhas-los na sequência preferida de desenvolvimento. Os desenvolvedores escrever em cada cartão uma estimativa de quanto tempo vai demorar para produzir a função; o utilizador patrocinador ou embaixador coloca-los em seqüência de prioridade de desenvolvimento, tendo em conta o tempo de desenvolvimento e valor comercial de cada função. Os cartões estão agrupadas em iterações, e essas iterações estão agrupadas em lançamentos, geralmente não mais do que alguns meses cada. O jogo de planificação é descrito em detalhes em (Beck, 2001)

O seguinte descreve a variação Eu gosto de usar. Eu chamo-lhe *Planificação Blitz* para enfatizar que ele vai rápido. No entanto, eu também gostaria de se referir a ele na ocasião como um *Planificação de projeto "Jam Session"* para enfatizar que todo mundo é suposto a jogar juntos e longe um do outro, de forma colaborativa, como em uma sessão de jazz jam. Se você perder a colaboração amigável, você pode fazer a técnica, mas você perde muitas oportunidades para melhorar criativamente o plano do projeto.

Acho que esta técnica funciona bem para um horizonte de planificação até cerca de três meses. Depois disso, a quantidade de detalhes é impressionante. Para horizontes mais longos de tempo de três meses, eu uso o *Mapa projeto* descrito entre os produtos de trabalho 22.

Aqui está um breve resumo da técnica. I descrever as etapas em mais detalhes abaixo. Reunir em um único representantes dos quartos de cada categoria das partes interessadas, o *Patrocinador executivo,* utilizadores finais e desenvolvedores, em particular. Brainstorm em cartões de índice de todas as tarefas de desenvolvimento a ser feito. Agrupar e mesclar as tarefas para evitar a duplicação. Sequenciar os cartões de acordo com a prioridade e da ordem de dependência. Neste ponto, os desenvolvedores colocar para baixo as estimativas de tempo de trabalho. Se uma pessoa em particular é necessário para a tarefa, adicionar o nome dessa pessoa. Se há uma dependência em um grupo externo, escreve que em outro lugar no cartão.

Ter as cartas na mesa, trabalhar através do plano de procura de marcos importantes e otimizações. Identifique o *Caminhando de esqueleto,* a primeira entrega, e a primeira entrega que produz um fluxo de receita. Procure dependências excessiva de qualquer pessoa, e off-load essa pessoa. Olhe para as tarefas iniciais que desnecessariamente bloquear fluxo de receita mais cedo, e as tarefas listadas para o final de que para o risco razões de redução deve ser feito

---

22 Este é um exemplo de planificação do projeto "de duas camadas". O plano de longo prazo é a coarse- grão *Projeto Mapa,* o plano de curto prazo é o resultado da *Planificação Blitz* sessão. Dois níveis planos de projeto são comuns em projetos ágeis.

(Por exemplo, testes de carga) cedo. Mova os cartões de volta para lugares melhores. Trabalhar com o patrocinador para garantir que o plano é entregar a funcionalidade de acordo com as verdadeiras prioridades do projeto.

O resultado é o plano do projeto.

Figura xx.xx mostra um exemplo de um tal cartão, e Figura xx.xx mostra um exemplo de como os cartões de olhar quando colocado sobre a mesa.

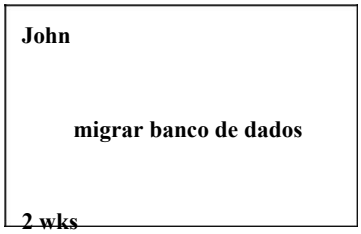


Figura 3-9. Amostra cartão de planificação Blitz.

Aqui estão os passos em detalhe. Note-se que esta técnica é soletrado para fora para o nível 1-praticantes 23. Quando você atingir o Nível 2 com esta técnica, tente algumas variações para lidar com situações em que os requisitos ainda não são conhecidos, aonde o horizonte de tempo é mais de três meses, utilizando notas em vez de cartões, e assim por diante.

1. Reúna os participantes

I legendar esta técnica de elaboração de projetos "jam session", porque a técnica permite que as diferentes partes interessadas no projeto para reunir e compartilhar ideias sobre como fazer o plano ideal. Tradicionalmente, o plano é feito pelo gerente de projeto ou líder da equipas, sem buy-in a partir das outras partes interessadas. Isso leva a duas disfunções: Primeiro, o plano é, naturalmente errada, uma vez que a pessoa pobre atribuído a construí-la não pode saber todas as tarefas e tempos; segundo, quando os erros no plano de projeto tornam-se evidentes, as pessoas acham fácil apontar dedos acusadores para a pessoa que fez o plano. Para contrariar estas disfunções, verifique se o *Patrocinador executivo,* um ou dois utilizadores-chave, nenhum analistas de negócios e toda a equipas de desenvolvimento, incluindo qualquer pessoa envolvida em testes e implantação, estão na sala. Este grupo irá nomear as negociações de forma mais completa, as estimativas de tempo será mais razoável, e tão importante, todo mundo vai ver todas as Trocas-offs feitos na construção do plano. Todo mundo é co-responsável pelo resultado, e todos eles sabem disso. Como uma pessoa disse: "Nós todos feitos, todos nós discutimos as otimizações feitas."

2. Brainstorm as tarefas

Todo mundo pega cartas e escreve tarefas neles o mais rápido que puder. Normalmente os desenvolvedores de fazer a maior parte da escrita, mas o patrocinador executivo e embaixador

23 Consulte "shu-ha-ri" nos primeiros e-mails, página 25.

user muitas vezes têm algumas tarefas para contribuir. Liste todas as tarefas que terá metade de um dia a várias semanas, incluindo entrevistas com utilizadores, programando funções específicas, escrever um texto de ajuda do utilizador, migrando os bancos de dados, e instalação do sistema (muitas vezes minúsculo grupo tarefas juntos). A idéia é ser completa. Esta etapa pode durar 5, 10, ou 15 minutos.

3. Separe as tarefas

Todo mundo coloca as cartas sobre a mesa, a fim dependência, os primeiros na cabeceira da mesa, com tarefas sucessivas para baixo da mesa. As tarefas que podem ser executadas em paralelo são colocados lado a lado por cima da mesa; tarefas que são executadas sequencialmente estão colocados acima e abaixo um do outro; duplicados cartões de tarefas são removidos.

Ele geralmente requer uma grande mesa. Eu tenho usado uma mesa de conferência grande e final também várias mesas do refeitório seis pés colocado ao fim.

4. Reveja as tarefas

Todo mundo anda para cima e para baixo da mesa, olhando para tarefas que não se capturados no brainstorming. cartão de uma pessoa muitas vezes desencadeia um pensamento em outra pessoa. Adicionar cartões, conforme necessário.

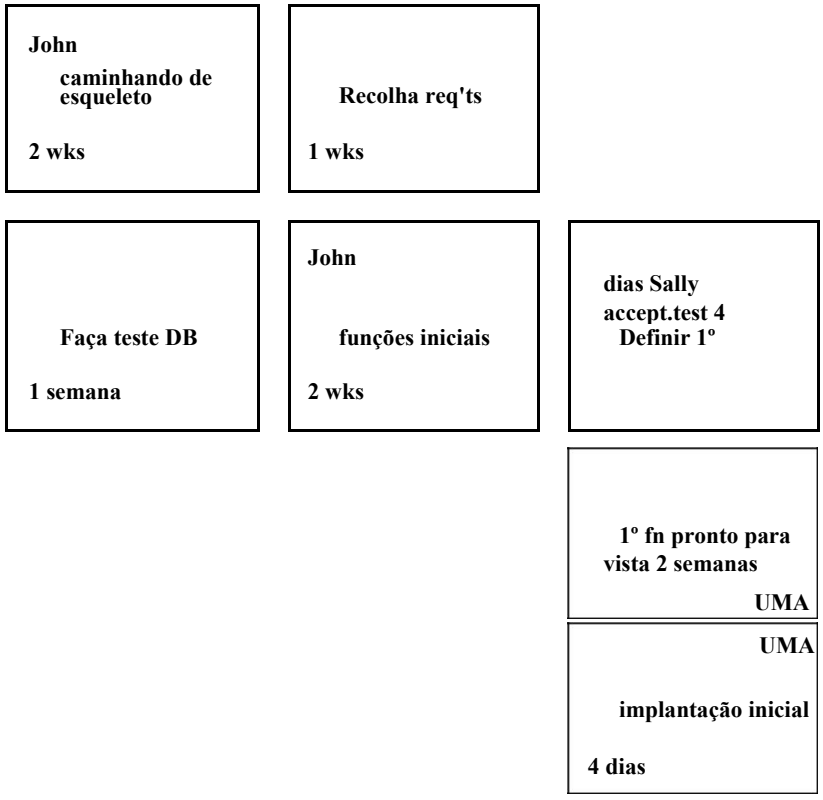


Figura 3-10. Cartões como eles podem ser colocados em cima da mesa.

5. Estimar e marcar as tarefas.

Quem vai fazer cada tarefa escreve para baixo sua estimativa de quanto tempo levará (eu escrevo isso no canto inferior esquerdo). Se várias pessoas vêm com diferentes estimativas, eles têm uma breve discussão e colocar o seu melhor segunda estimativa. Eles também anotar os nomes de quaisquer pessoas específicas necessárias para qualquer tarefa particular (eu escrevo isso no topo do cartão). Muitas vezes acontece que o nome do líder da equipas está em um número desproporcional de cartões. Vendo o nome dessa pessoa em tantos cartões devem liderar a equipas em uma discussão de como off- carregar essa pessoa ou de outra forma reduzir a dependência sobre ele. Se uma tarefa está dependente de um evento externo ou tarefa, marcar esse no cartão (eu escrevo isso no centro do lado direito do cartão.)

6. Classificar as tarefas

Nesta etapa, as pessoas trabalham para identificar mais de perto a presença ou ausência de dependências, e questionar a colocação de cartas na mesa.

Tarefas que são colocados sequencialmente muitas vezes pode ser iniciada em paralelo. Identificar possível paralelismo afrouxa as restrições sobre o plano. Além disso, muitas vezes acontece que existem fluxos paralelos de atividade, um para o *Embaixador do utilizador,* uma para os desenvolvedores, ou talvez três correntes separadas para infra-estrutura, funcionalidade e desenvolvimento de interface de utilizador.

Estes fluxos paralelos começar a olhar como faixas separadas de cartões que funcionam abaixo da tabela, ocasionalmente, se unindo.

Alguns cartões têm uma relação estritamente sequencial, de tal forma que a segunda simplesmente não pode ser iniciado antes do primeiro ter terminado (exemplo: "Avaliar fornecedores" deve preceder totalmente "Configurar contrato com o fornecedor"). Marque esses pares de alguma forma especial para que a informação não se perde quando os cartões são movidos em torno de (I colocar a mesma letra maiúscula, A, B, C, em uma borda adjacente do cartão, a borda inferior do primeiro cartão e o ponto correspondente na aresta superior da segunda placa). Estas dependências são rigorosos surpreendentemente raro; Eu raramente ir além da letra E.

7. Marque o *Caminhando de esqueleto,* a versão mais antiga e a mais antiga receita

Encontrar o primeiro e menor coleção de funcionalidade que pode concebivelmente ser de utilidade para alguns utilizadores é fundamental. É fundamental, porque esse lançamento representa a primeira vez que os utilizadores verão o sistema, a sua primeira ideia do olhar, sentir e forma do sistema. Definir esse ponto tão cedo quanto possivelmente dá toda a organização mais tempo para se adaptar a todas as ideias novas ou desencontros que aparecem a partir do lançamento inicial. Em algumas ocasiões, este primeiro lançamento representa o mais cedo em que a receita pode ser adquirida; nesta situação, o grupo deve se concentrar em obter todas as outras cartas fora do seu caminho 24. Ao todo, três eventos de

24 *Software por números (* OMS, 2004 ???) é um excelente tutorial sobre o assunto, usando uma linguagem de analistas financeiros.

Eles planejam em termos de "mínimos recursos negociáveis" (MMF).

interesse estão localizados: o *Caminhando de esqueleto,* o lançamento utilizável mínimo, e o ponto de primeira receita.

Eu marcar esses pontos, colocando um pouco de distância entre os cartões acima e abaixo desses pontos. Eu usei também corda, bitolas, lápis - qualquer coisa para demarcar os pontos.

Ela requer a cooperação entre a *Embaixador do utilizador, o Patrocinador Executivo,* e *Designer-chefe* para descobrir e otimamente definir esses pontos, porque algumas tarefas terão de ser movida para cima e outros adiada. Quem está se movendo as tarefas em torno precisa estar ciente das consequências destes movimentos, e isso significa que cada grupo de interessados tem de estar presente, alerta e cooperativa.

A palavra está em ordem sobre as tarefas que precedem o lançamento inicial: Alguns dos primeiros tarefas são susceptíveis de ser estritamente técnico, como encomendar software, a criação de um contrato, ou carregar um banco de dados. É útil ter essas tarefas visíveis a *Patrocinador executivo* e *Embaixador do utilizador,* porque eles precisam explicar o estado do projeto para outras pessoas. Os cartões de tarefas técnicos mostram que o trabalho precisa ser feito antes de software de negócios se tornará visível.

Historicamente em nosso negócio, o grupo de utilizador recebe apenas disse: "A equipas de programação diz que eles têm um monte de trabalho técnico que fazer primeiro." Isso muitas vezes se traduz em meses que passam sem entregas ou mesmo progresso visível. Ao fazer as tarefas visível e público, o *Patrocinador executivo* ou *Embaixador Utilizador*

pode relatar: "Eles têm cinco coisas técnicas para fazer antes de podermos ver alguns software," "Eles ainda têm duas coisas técnicas para fazer antes de podermos ver alguns software", "Eles estão na última coisa técnica, e então eles' vai nos mostrar algum software em execução."

Esse tipo de visibilidade vai um longo caminho para reduzir as tensões entre os grupos.

8. Identificar outros lançamentos

O grupo trabalha para fora aonde outros pontos de lançamento naturais acontecer. Muitas vezes, há um conjunto particular de função que realmente devem ser agrupados para os utilizadores a fazer bom uso do sistema. Em algum momento, as funções a serem incluídos simplesmente tornar-se uma longa lista, e nesse ponto ele pode ser mais valioso para fundamentar os lançamentos em períodos regulares (mensais, bimestrais ou trimestrais), em vez de coleções de funcionalidade.

Nem Crystal Clear nem a técnica mandato Blitz Planificação qual algoritmo que usa para escolher comprimentos de iteração e liberação períodos.

Marque as liberações chaves utilizando um espaço entre as placas, fios, bitolas, ou o que quer.

9. Otimizar o plano para atender as prioridades do projeto

Neste ponto, você tem um plano. Típico o plano ainda não é um *Boa* plano. A minha experiência é que quando os números sobre os cartões são somados, a equipas está em para "choque da etiqueta" (este é o choque de um comprador do carro experimenta ao pisar em torno de ler o adesivo afixado na janela do carro anunciando o preço depois de tudo opções do carro são somados).

todo o grupo, e particularmente o *Patrocinador Executivo, User Embaixador* e *Designer-chefe* agora ter alguma resolução criativa de problemas que fazer para chegar a um plano aceitável. O que eles mudar depende das prioridades definidas para o projeto: time-to-market, custo ou conjunto de recursos. Este é o lugar aonde eles realmente começar a "interferência". Normalmente, após o choque, uma das três coisas acontecem:

- **o *Patrocinador executivo* reconsidere a necessidade do negócio para o projeto (em um caso, a equipas executiva decidiu simplesmente comprar um pacote comercial vez);**
- **A equipas remove tarefas do projeto;**
- **Eles dão de ombros e simplesmente seguir em frente. Pressionar os desenvolvedores a mudar suas estimativas não é aconselhável. Isso não só torna uma mentira do plano, mas convence os desenvolvedores que a *Patrocinador executivo* não é ser realista, aberto e honesto.**

No pior caso possível, o *Patrocinador executivo* tem o direito de dizer: "OK, eu vejo as tarefas e as estimativas de tempo. No entanto, não pode alterar o prazo, e eu não posso ver mais nada para remover, reordenar ou terceirizar. Vou multiplicar tudo estimativas de 80% para que possamos cumprir a nossa data-alvo. Nesse meio tempo, vamos todos manter nossos olhos abertos para novas alternativas para que possamos restabelecer estas estimativas originais"(ver a discussão sobre Crystal Clear e 'Realocação de Energia', na página 309)

Esta é uma medida drástica e não deve ser usado levianamente. O ponto de fazê-lo é tornar público, tanto como as prioridades afetar o cronograma, eo fato de que as estimativas da equipas estão sendo substituídas. Todo mundo já viu os cartões e as estimativas originais. A equipas deve monitorar ambos os planos, relatório contra ambos em suas tabelas de status e manter a visibilidade do progresso alta.

Levá-la a sério, se você tem que recorrer a esta medida até duas vezes. Isso indica que há um outro problema na organização a necessidade de ser resolvido, tendo a ver com a viabilidade do modelo de negócio da organização, a confiança entre os desenvolvedores e os patrocinadores, ou a precisão histórica de estimativas da equipas de desenvolvimento. Endireitar aqueles fora será crucial para a viabilidade contínua da equipas de desenvolvimento.

Ao otimizar o layout do cartão sobre a mesa, as pessoas olham para três melhorias, em particular.

- **Eles local que um grupo particular de cartões *quase* cria uma versão coerente com o valor do negócio. Ao mover um cartão mais alto, eles podem obter esse valor de negócio coerente muito mais cedo. Em um caso, ouvi o *Embaixador Utilizador* dizer: "Se**

nós só tinha este cartão até aqui, poderíamos começar a cobrar as receitas com esta versão." Você pode adivinhar o que se imediatamente propôs que o cartão para cima, *e em seguida, mudou-se todos os cartões não sejam estritamente necessários para produzir esse lançamento!*

- **Eles local um risco de ter um cartão no final do projeto. Isso geralmente é feito por alguém com formação técnica jogando o desafiante.**

Eu vi uma placa perto do fim da tabela rotulada "Load Testing." Não pensar muito sobre isso, pedi ao *Designer-chefe:*

"Quanto tempo vai demorar para executar? "Ele respondeu: 'A poucos dias.' Ainda não se preocupar, eu perguntei, 'E suponho que não, quanto tempo é provável que tomar para rever a arquitetura?' Ele respondeu:" Oh, três ou quatro semanas." neste ponto, o *Embaixador Utilizador* começou a olhar alarmado. Eu perguntei, "seria possível fazê-lo mais cedo?" "Claro", respondeu ele.

Nós procuramos o mais cedo em que ele poderia fazê-lo sem danificar os outros itens críticos de negócios em cima da mesa. Eu tentativamente colocou-o logo após o lançamento inicial. "Você poderia fazê-lo já aqui? (E então você tem muito tempo para rever a arquitetura se ele falhar)" Eu observei o *Embaixador Utilizador* veementemente balançando a cabeça do outro lado da mesa, e notou seu grande suspiro de alívio quando ele disse: "Claro."

- **Eles local que uma pessoa em particular tem muitos cartões alocados para ele ou ela. Essa pessoa vai ser sobrecarregado e provavelmente vai ter problemas. Eles debater como atribuir essas tarefas, ou a maior parte de alguns deles, para outras pessoas para reduzir o risco para o projeto e a tensão sobre essa pessoa.**

10. Capturar a saída.

Você tem um plano, mas é apenas cartões sobre uma mesa. Você precisa preservar as informações.

Você pode fotografar a mesa, fita as cartas juntos e montar o na parede como um radiador de informações, ou digite-os em outra ferramenta de sua escolha.

Eu gosto de numerar os cartões de modo a que a sua colocação sobre a mesa podem ser reconstruídos. I numerá-las 1, 2, subsequï¿½ncias 3, e assim por diante até a mesa para o relações sequenciais, 3a, 3b, 3c e assim por diante em qualquer conjunto paralelo particular (e na ocasião 3a1, 3a2, 3a3 e assim por diante se não estão aninhadas).

*** * ***

Há uma nota que eu sou obrigado a colocar aqui e repetir várias vezes no livro. A interpretação comum do jogo de planificação tanto do XP e *Planificação Blitz* é esse o

***Patrocinador executivo* está à mercê dos desenvolvedores na construção do plano do projeto. Na realidade, há uma divisão de três vias de responsabilidades: o *do Patrocinador Executivo,* os desenvolvedores, e sua responsabilidade conjunta.**

o *Patrocinador executivo* é responsável por escolher as prioridades que orientam qual sistema características ficar sobre a mesa, que são removidos se o escopo é para ser cortado, e que funções devem ir para quais liberação.

Os desenvolvedores são responsáveis por avaliar o tempo necessário para fazer o seu trabalho. o *Patrocinador executivo* precisa reconhecer que estas são as pessoas que ela contratados para o trabalho; eles são profissionais; estes são os seus melhores estimativas.

Eles são co-responsáveis por ser criativo na criação de estratégias para maximizar a sua eficácia. Eu costumo encontrar o layout inicial das cartas na mesa deprimente, e insatisfatório por razões comerciais. Os desenvolvedores não podem alterar as prioridades e a *Patrocinador executivo* não pode alterar as estimativas. O que eles estão a fazer?

Eles trabalham juntos. A disposição inicial é geralmente ineficiente. Trabalhando juntos, eles vêm com reordenações criativas para obter um melhor resultado. Se o plano resultante ainda é inaceitável, o *Patrocinador executivo* deve decidir o que os itens de prioridade de topo são, o que obtém caiu, ou talvez se o prazo ou a equipas ser alterado.

Quando eles trabalham juntos, eles vêem em conjunto as restrições, gerar conjuntamente opções, e construir em conjunto um plano que produz o melhor resultado para os recursos gastos. É por esta razão que eu às vezes se referem a esta técnica como o planificação do projeto *jam session.*
(Sl Feito com boa vontade, essa técnica também é muito divertido, outra característica de um bom *jam session.)*

Técnica 4. **Delphi Estimativa usando a perícia Rankings**

As pessoas sempre dar a volta a perguntar: "Mas como devemos estimar o período de tempo que vai demorar para desenvolver o software?" A verdadeira resposta é, "Melhor palpite." Embora tecnicamente correto, e praticada em cada projeto, esta resposta raramente é reconfortante.

Um grupo de nós criou uma técnica de estimação uma noite, enquanto tentando descobrir o lance em uma $ 15 milhão de preço fixo, projeto de escopo fixo. Tenho desde a linha de raciocínio que levou a ter uma separação interessante de perguntas. Disseram-me que isto corresponde ao que é conhecido como a técnica "Delphi" (prever o futuro, mas presumivelmente sem incenso, virgens vestais, ou atender em enigmas).

Esta técnica foi descrita em primeiro *Sobrevivendo Projetos Orientados a Objetos.* Eu apresento-lo aqui, em forma de história. A história diz respeito a um projeto maior Crystal Orange, mas você ainda pode usá-lo de forma ajustada em um projeto menor.

> **Antes de iniciar o projeto adequado, mas depois de passar duas semanas reunindo 140 casos de uso ásperas, nós fizemos nossa primeira estimativa projeto. Nossos quatro melhores designers OO, o gerente do projeto, e alguns outros experientes, pessoas não-OO compunham a equipas de planificação. Nós dividimos a sessão em quatro fases:**
>
> **- estimar o tamanho do sistema a ser construído**
>
> **- estimar o tempo de trabalho de acordo com o tipo de pessoa que seria necessário**
>
> **- sugerindo releases, por dependência técnica e de negócios**
>
> **- equilibrando as versões em tamanhos aproximadamente semelhantes. Nós realizou um leilão aberto para chegar a uma estimativa de tamanho. Construímos uma grande mesa no quadro branco. Cada designer sênior criou linha de rótulos para os fatores que ele pensou que iria determinar o esforço do projeto. Um deles escreveu: "quadros técnicos, casos de uso, telas de interface do utilizador." Outro acrescentou: "classes de negócio"; outro acrescentou: "ferramenta de geração de banco de dados".**
>
> **Cada pessoa escreveu em sua coluna da tabela o seu palpite de quantos de cada fator estava presente. Depois que todos tinham tido a sua vez, fizemos uma segunda rodada. A primeira pessoa adicionada uma nova coluna com sua nova estimativa com base no que ele tinha aprendido durante a primeira rodada.**
>
> **Fizemos três rodadas desta forma. No final, havia cerca de 20-25 factores em todos. Algumas pessoas usados factores de multiplicação a partir da estimativa de classes de negócio, alguns dos casos de uso, algumas das telas da interface do utilizador. Descobriu-se que os factores-chave para as estimativas foram o número de:**
>
> **- classes de negócio**
>
> **- telas**
>
> **- enquadramentos**
>
> **- aulas técnicas (infra-estrutura, serviços públicos, etc.)**

Discutimos se tivéssemos conseguido a convergência, e quais são os fatores diferentes eram. No final, nós concordamos com alguns números e compreendido em que diferiam.

Na segunda fase, decidimos que tipo de pessoa que seria necessário para cada seção do código. Frameworks são difíceis, e há apenas um seleto número de pessoas que podem escrever-los razoavelmente, então fizemos tão sensível para a pessoa específica que poderia contratar. Decidimos que as classes de negócio seria relativamente fácil de escrever, mas exigiria conhecimento do negócio. aulas técnicas seria mais difícil, mas não exige conhecimento do negócio. No final, estamos liquidados em:

- **desenvolvedores especializados para os quadros**
- **desenvolvedores de nível intermediário para aulas técnicas e de interface do utilizador**
- **desenvolvedores relativamente novatos que conheciam o domínio para essas classes Nós dividir a tabela de classes a serem desenvolvidas para essas três categorias, resumiu as classes por categoria, e decidiu sobre quantas aulas por mês esse nível de desenvolvedor poderia desenvolver. Demos os desenvolvedores quadro 10 semanas por quadro, os desenvolvedores técnicos e UI 3 semanas por turma, e os desenvolvedores de classe empresarial 2 semanas por classe.**

A soma de todas essas semanas deu-nos a nossa estimativa de tempo primeiro projeto. Nós cercado e hawed sobre isso por um longo tempo, comparando-a com outras medidas de razoabilidade, tais como a taxa poderíamos contratar e treinar pessoas. No final, nós mantivemos a estimativa.

Decidir os lançamentos foi simples. Nós já sabíamos que queríamos um lançamento a cada três meses. Nada poderia ser entregue até que a maioria da infra-estrutura foi feito, de modo que entrou em liberação 1 [***Nota:*** **que mais tarde foi completamente substituído, usando o** ***rearquitetura incremental*** **estratégia]. Nós criamos um** ***Projeto Mapa,*** **um gráfico de dependência das funções de negócio, por dependência técnica e de negócios. A partir das estimativas de tamanho, que circulou áreas de tamanho similar. Isso deu-nos o nosso plano de lançamento.**

Eu segui o projeto contra a nossa estimativa original periodicamente durante o projeto. Nosso progresso acompanhado nosso plano original, a longo prazo, mas indo mais lento no início e depois mais rápido no final.

Eu apresento esta técnica aqui porque eu noto que a maioria das pessoas que usam uma técnica Delphi esquecer a segunda fase crucial, quando o grupo avalia a sua capacidade de contratar pessoas específicas com talentos e habilidades específicas. Sem esta informação o resultado não é um plano, mas um desejo.

Técnica 5. **Reuniões diárias de stand-up**

A reunião diária stand-up é uma breve reunião para trocar notas sobre status, progresso e problemas. A palavra-chave é curto. A reunião não é usado para *discutir* problemas, mas para *identificar* problemas. Se você achar que você está discutindo como resolver problemas em seu stand-up diário, levante a mão e pedir que a resolução do problema ser tratado logo após a reunião stand-up, com apenas as pessoas que têm de estar lá.

O termo "stand-up diário" vem da metodologia Scrum (Schwaber 2002). A idéia é livrar-se de longas chamadas reuniões de status aonde os programadores divagar sobre por cinco ou dez minutos sobre um pedaço de código que eles estão trabalhando, ou gestão de pessoas divagar sobre várias iniciativas de projetos. Para manter as pessoas de caminhadas, a convenção é que a reunião tem lugar em pé, para que as pessoas não podem cair no sono, tipo em seus laptops ou rabiscar no papel. Queremos que eles sensíveis à passagem do tempo.

Os autores do Scrum sugere que cada pessoa simplesmente responde a três perguntas:

- **O que eu trabalho ontem?**
- **O que eu pretendo trabalhar hoje?**
- **O que está ficando no meu caminho?**

A reunião diária stand-up é surpreendentemente eficaz para a divulgação de informações, destacando quando alguém é preso, revelando quando o item que alguém está trabalhando é muito baixa prioridade, é off-topic ou adicionar funcionalidades não solicitadas, e geralmente manter o grupo focado e em pista. É simples, e foi adicionado em projetos usando cada metodologia imagináveis com um bom efeito.

Técnica 6. **Interaction Design Essencial**

A maioria dos autores sobre o desenvolvimento ágil não dizem muito sobre design de interface de utilizador 25, dando a impressão de que acho que não é importante. Falando pelo menos para mim, eu não escrever sobre isso porque eu tenho tão pouca experiência pessoal com esta especialidade, apesar de eu reconhecer sua importância.

Por essa razão, eu incluo aqui quatro adaptações de design centrado no uso (Constantine 1999 ???) que Jeff Patton criado para o contexto ágil. Jeff destaca e simplifica as atividades necessárias para definir, design e testar o sistema de *contextos de interação* e os seus *personalidades 26.* Estas técnicas são bem adequados para projetos aonde as pessoas são co-instalados e tem que ter um monte realizado em um tempo limitado. (Só porque você tem Fácil acesso a utilizadores experientes não significa que você começa a perder o seu tempo!)

Sobre personalidades. **Um sistema de software apresenta uma certa quantidade de ajuda, informação e velocidade para seus vários utilizadores finais. Pode ser projetado para ser rápido e eficiente, por exemplo, ou acolhedor, simpático e informativo. Estes são o *personalidades* ele apresenta para o mundo exterior. A maioria dos sistemas apresentam diferentes personalidades para utilizadores diferentes, dependendo de fundos e necessidades desses utilizadores. Exceto . . . a maioria dos designers não desenvolvem as personalidades de qualquer forma deliberada. Eles simplesmente jogar juntos tudo o que tem na mão, e as personalidades do sistema são apenas o que acontece para sair.**

Há provavelmente um ou dois *focal* grupos de utilizadores que os patrocinadores querem realmente ver satisfeito com o novo sistema. A equipas precisa descobrir quem esses são, o que eles precisam para realizar usando o software, o que a personalidade é mais adequado para cada um, e ter certeza que eles estão devidamente servidos pelo sistema. Eles precisam de detectar, escrever, e periodicamente verificar novamente que esses são.

Jeff ilustra com o exemplo de uma cadeia de lojas de varejo. O caixa usa o sistema diariamente, fica familiarizado com ele, e prioriza a velocidade em registrar vendas. O consultor loja, por outro lado, não vai usar o sistema, muitas vezes, não vai ser fluente com sua interface, mas sabendo que todos os aspectos das operações da loja, vai querer fazer mais funções do que o caixa. O caixa quer uma personalidade rápido e eficiente para trabalhar com; o consultor loja quer um lugar informativo-and-útil.

Sobre contextos de interação. **Ao trabalhar através de suas tarefas, os utilizadores vão querer ver informações em clusters. Parte do design de interação está detectando o agrupamento de informações adequadas para tarefas de cada um dos utilizadores, e à procura de uniformização ao nível daqueles. Os desenvolvedores irá converter aqueles a ajustes de interface do utilizador, muitas vezes com um conjunto de informações correspondentes de um contexto de interação.**

---

25 (Cohn 2004) rompe o silêncio com um capítulo sobre design centrado no uso.

26 Meu prazo. Parece-me a comunidade de design de interação está faltando este conceito de nível superior do seu trabalho.

***Interaction Design Essencial* produz**

- **entendimento compartilhado entre os patrocinadores, utilizadores e desenvolvedores que estavam presentes na sala como para as funções e tarefas a serem abordados pelo sistema e as prioridades relativas de cada um, e também**

- **colagens com papel e caneta ("lembrando marcadores" na linguagem cooperativa-game), mostrando funções, tarefas e contextos de interação, marcados com notas de modo que a equipas possa entregá-los, a fim valor do negócio. Jeff articula quatro técnicas, em todos.**

1. **Interaction Design Essencial (Workshop)**
2. **Derivando a UI**
3. **Inspeção de Usabilidade (durante o projeto)**
4. **QA Testing as Personalidades do sistema**

O primeiro é design de interação essencial em si, feito quer perto do início do projeto ou o início de uma iteração. Depois disso, ele apresenta técnicas de curta duração para projetar as telas de si mesmos, para rever a interface do utilizador com os utilizadores, e para testar internamente que o software acabado é apropriado para as funções de utilizador que irão utilizá-lo.

Aqui estão as técnicas como Jeff descreve-los.

*** * ***

Interaction Design Essencial (Workshop)

A abordagem baseia-se na abordagem de design utilização centrada (Constantino 1,999 ???), excepto que este elicitação requisitos iniciais e processo de concepção pode ser completada em um par de horas, para um par de dias. Quando um facilitador experiente é usada, nenhuma formação avançada dos empresários é necessária. As técnicas podem ser aplicadas durante requisitos iniciais elicitação, quando a construção de planos de lançamento incrementais, ao definir a forma geral do software, derivando interface de utilizador de casos de uso, testes de aceitação e até mesmo durante a avaliação de usabilidade do utilizador final, se for necessário.

O que se segue é um passo a descrição passo do processo de elicitação de requisitos e projeto inicial. O objetivo geral da suíte técnica é *acelerar a descoberta de requisitos, reunindo pessoas de cada aspecto crítico do projeto e permitindo-lhes para explicar o que eles sabem uns dos outros.*

Passo 1. Obter as pessoas certas para o quarto.

Estamos substituindo semanas de entrevistas e pesquisa de campo com conversa durante esta reunião, por isso é fundamental a reunião envolver membros de cada circunscrição o sistema estará servindo, inclusive o desenvolvimento do sistema. Lembre-se, se você é o facilitador, você não está interessado no que os participantes dizem para você, mas o que dizer um ao outro.

Você vai precisar na oficina

- **seleccionadas as principais partes interessadas do projeto,**

- **utilizadores-chave selecionadas,**
- **especialistas de domínio, e**
- **membros da equipas de desenvolvimento, incluindo test / QA pessoas. Oito a doze pessoas são**

boas. Às vezes é difícil para limitar os participantes a esse número, mas fazê-lo.

Definir os participantes em um espaço de trabalho confortável com uma grande mesa de trabalho, os lotes de espaço na parede para pendurar cartazes. Incluir marcadores, fita adesiva, 3 "x 5" cartões de índice e lotes de lanches para manter as pessoas menos ocupados ocupados.

Você está hospedando um partido.

Passo 2. Captura e anotar as funções do utilizador.

O termo "agente" refere-se a qualquer de cargo, função de utilizador, ou uma mistura de ambos. Estamos atrás de "papéis", frases que indicam quais são os objetivos específicos dos utilizadores. Eu sempre olhar para frases "coisa doer" para descrever um papel, ou mesmo " *adjectived* coisa doer ", para torná-la mais específica (somente em Inglês pode um substantivo descrevendo um adjetivo ser usado como um verbo no tempo passado!).

Por exemplo: podemos encontrar não apenas um "fim-taker" na situação loja, mas um "tomador de ordem especial", ou melhor ainda, um Para lidar com clientes irritados porque a ordem especial é "apressada tomador de ordem especial." tarde, encontramos um "pesquisador de ordem tarde" ou um "manipulador de queixa." (as pessoas com títulos como Associado de Vendas e Gerente de executar todas as funções acima.) o nome da função estendida torna mais fácil para manter o papel claro o suficiente para que qualquer pessoa , com ou sem conhecimento de domínio especial, pode entender o que o papel poderia estar fazendo. Ele nos permite saber a pessoa que executa esse papel está com pressa e precisa de funcionalidade, e uma interface de utilizador, que pode suportar isso.

funções de utilizador Brainstorm para os cartões de índice "cartão-storming", como Constantine e Lockwood chamá-lo. Após cartão-storming, refinar sua lista de papel, removendo duplicatas e verificar os nomes são claras. Bons nomes são importantes. Não vamos estar contando com páginas de documentação para descrever cada função, por isso o nome é tudo que você tem. Certifique-se de que é expressivo.

É **provável que você nomear os papéis que você pode considerar os intervenientes do sistema, não necessariamente utilizadores. Não descarte esses. É importante considerar os objetivos dessas pessoas. É importante perguntar como o software poderia apoiar essas metas. Eu sempre acho que os papéis que poderiam ter sido postas de lado como partes interessadas realmente precisa funcionalidade do sistema que suporta provando a eles que seus interesses eram protegidos.**

No cartão de índice, escreva o principal objetivo do papel ou metas sob o nome da função. O que constitui o sucesso para esse utilizador no uso do sistema? O que constitui o fracasso? Vamos manter a verificação de que os requisitos que capturar realmente ajudar as funções do utilizador atingir suas metas.

Figura 3-11. A sessão de modelagem papel. Certifique-se de ouvir as conversas que ocorrem durante este processo. (Graças a Jeff Patton)

Escrever apenas o que constitui o sucesso do ponto de vista da função de utilizador, não do seu chefe ou alguma outra das partes interessadas do. Por exemplo, um funcionário do call center em um centro de serviço de cartão de crédito, uma "aplicação tomador de cartão de crédito-" pode ter uma meta para levar aplicações com rapidez suficiente e com precisão suficiente para evitar a atenção de um gerente. O gerente, por outro lado, pode querer melhorar a velocidade e precisão e reduzir as reclamações dos clientes. Outras partes interessadas podem estar preocupados com as políticas de seguro de crédito up-selling. O gerente de e objetivos das partes interessadas não são necessariamente as do "tomador de aplicação de cartão de crédito." Certifique-se que o objetivo apropriado encontra o seu caminho para o cartão de papel apropriado. Se você deseja capturar as necessidades dos demais stakeholders, acrescentando considerar alguns papéis 'observador de eficiência', 'upselling observador,' e criar papel cartões para eles.

Escrever em cada cartão uma estimativa subjetiva do valor de negócios que de papel ou importância, alto, médio ou baixo. Eu marcar "H", "M", ou "L" no canto inferior esquerdo do cartão.

Escrever em cada cartão uma estimativa de quantas vezes o papel vai usar o sistema. Isso pode ser alta, média, baixa ou horária, diária, mensal, trimestral, anual, ou alguma outra frequência que achar adequado. Escrevo estas no canto inferior direito do cartão.

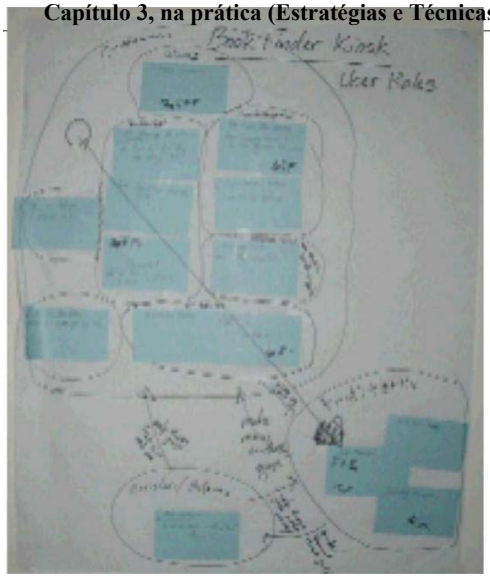
Passo 3. Construir o Modelo de função do utilizador

Nesta etapa, você entender e marcar as relações, dependências e colaborações entre papéis. O resultado é a versão ágil de um *modelo de papel.*

Usando uma folha de papel cartaz, os participantes irão agrupar os cartões de papel do utilizador na tabela. A única instrução importante dar é: "Tente cluster. Mantenha papéis semelhantes entre si juntos, papéis diferentes mais distantes." Uma vez que o modelo "sente" certo, corrigir as cartas para o papel de cartaz com fita adesiva.

Os papéis provavelmente se agrupados porque eles têm objetivos semelhantes ou participar no processo de negócio, às vezes semelhantes. Circundam cada conjunto de funções. Rotular cada um com alguma indicação de por que esses papéis foi agrupado.

Desenhar uma linha de cada conjunto para outros conjuntos em que existe uma relação. Eu poderia, por exemplo, desenhar uma linha de um cluster marcado "ordem tomadores" a um marcado "cumpridores ordem" e rotulá-la "envia ordens para."



Faça quaisquer outras notas de interesse no modelo, possivelmente incluindo uma regra notável negócio, utilizador da função característica, ou relacionamento através funções de utilizador.

Figura 3-12. Uma amostra de papel-modelo. Para os presentes durante a sua criação, a folha marcada-up traz de volta uma enxurrada de conversa e informação. (Graças a Jeff Patton)

Passo 4. Identificar o *Focal* Roles.

Os participantes agora votar para os papéis (não grupos de funções!) Eles consideram ser mais crítico para o sucesso do software (incluindo também os papéis aonde conseqüências terríveis se seguem quando eles falham em seus objetivos). Eu normalmente só permitem que pessoas com qualquer experiência de domínio ou opiniões fortes e instruídos a votar.

Para obter os papéis focais, posso dar cinco eleitores 3 votos cada um. Desde há geralmente doces (por exemplo, os beijos de Hershey) sobre a mesa durante as nossas sessões, eu gosto de usar aqueles como fichas de voto. Ter pessoas colocar as fichas diretamente sobre os cartões de papel que eles consideram mais significativo. Fazer isso com marcadores físicos ao invés de apenas marcas de caneta é útil para as pessoas a fazer escolhas. Muitas vezes eu encontrar pessoas que já usaram-se os seus votos vai começar a pressionar os outros a votar como fizeram, ou votar para o papel que não o fez, porque eles acabaram de votos. Estes são conversas interessantes. Muitas vezes vejo talvez três ou cinco ou mais papéis escolhidos como focal de 15 a 25 de papéis sobre a mesa.

Depois de todos é feito, substituir os tokens físicos com marcas tais como um 'F' para cada voto. Escreva o 'F' usando uma caneta de cor viva. Isto torna mais fácil para encontrar os papéis focais em um modelo ocupado. Papéis com o mais 'F são fáceis de detectar. I lembrar povos que prestar muita atenção a estes.

Que são susceptíveis de fazer descobertas interessantes neste momento, por exemplo, que o mais votado em, ou focal, os papéis podem não ser os únicos com o maior valor de negócio ou ter a maior frequência de uso. Discutir seus resultados. Faça quaisquer notas interessantes diretamente no modelo.

Passo 5. Capturar as tarefas necessárias para cumprir as metas.

Usando o modelo como referência, pense em uma pessoa vai para o sistema para executar uma tarefa para cumprir a meta nesse papel particular. Escrever esse nome tarefa em um cartão de índice. Eu gosto de nomes de tarefas que começam com um verbo, como “adicionar itens à ordem” ou “escolher

cliente de clientes conhecidos."Assim como ocorre com nomes de função, os nomes de tarefas são importantes. Quanto mais clara e concisa os nomes tarefas, a documentação menos apoio que você precisa para eles.

"Cartão-tempestade" as tarefas de utilizadores que utilizam esta atividade de pensar, cobrindo todas as funções. Quando não é provável que seja um grande número de tarefas, eu gosto de trabalhar em um conjunto de papéis de cada vez. Remova duplicatas e esclarecer nomes de tarefas.

Para cada cartão de tarefa, anote diretamente no cartão de três coisas:

- **O objetivo desta tarefa. O que é um bom resultado para a tarefa?**
- **Sua frequência. Você pode usar alta, média ou baixa, ou referências de tempo, como por hora, diária, semanal, mensal ou anual.**
- **Seu valor comercial. Esta é uma suposição subjetiva melhor, alto, médio ou baixo.**

Passo 6. Construir o Modelo de Tarefa

Nesta etapa, você irá identificar relacionamentos e tarefas dependências têm uns com os outros, assim como você fez com o modelo.

Disponha os cartões de tarefa em uma folha de papel cartaz. Sem ter para coincidir com o agrupamento papel no modelo, aglomerar tarefas semelhantes e deixe tarefas diferentes caem distantes. Quando o modelo se sente bem, corrigir os cartões no lugar com fita adesiva.

Como com o modelo de papel, círculo e etiqueta aglomerados de cartões, em seguida, procurar relações entre clusters.

Desenhar e linhas de relação etiqueta entre os agrupamentos.

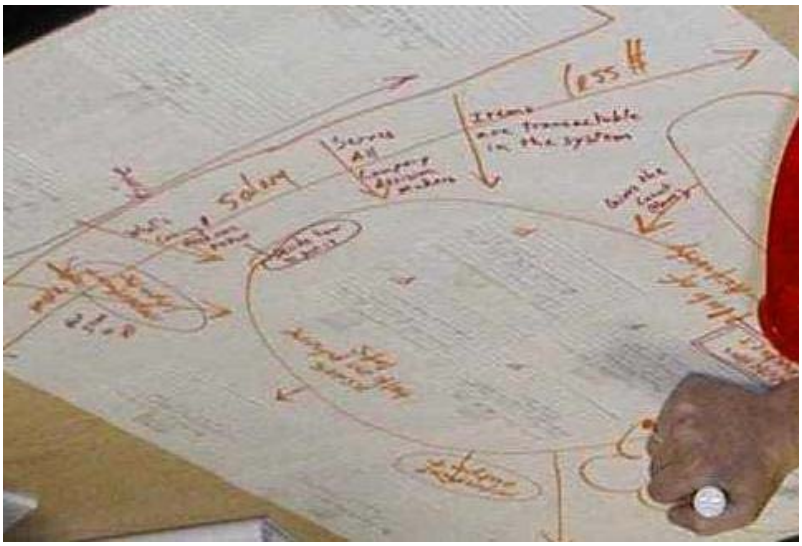


Figura 3-13. Marcando-se o modelo de tarefa. Os participantes muitas vezes inventar notação significativa para o domínio e para si mesmos. (Graças a Jeff Patton e Tomax)

Veja se você pode manchar uma 'linha do tempo."Olhe para a tarefa susceptível de ser realizado mais cedo. Comece lá e desenhar uma linha conectando cada tarefa posterior até chegar a tarefa propensos a ser realizada passado. Eu costumo encontrar a linha vai do lado superior esquerdo do modelo até o

inferior direito - mas não sempre. A linha do tempo ajuda a apontar o fluxo de trabalho global de tarefa em tarefa.

O modelo de tarefa deve ser uma imagem concisa do trabalho que o sistema precisa executar.

Passo 7. Identificar o *Focal* tarefas

Você precisa entender e marcar as tarefas que são mais críticos para o sucesso do software. Assim como com o modelo, todo mundo recebe três votos, ou talvez mais, se o modelo de tarefa tem um grande número de cartões.

Ter as pessoas votam para as tarefas mais importantes para o software. Que tarefas entregar o maior valor? Que tarefas são feitas com mais frequência? Alternativamente, pense nas tarefas que se não for feito, ou feito incorretamente pode causar mais problemas. Você pode usar o doce de novo se não tiver sido comido por agora.

Escrever um 'F' para cada voto diretamente sobre os cartões. Tal como acontece com o modelo de papel, use uma caneta coloridas para escrever os 'F de nos cartões. Esses são os seus *focal* tarefas.

Procure descobertas interessantes. Há tarefas de alta frequência com alto valor de negócio que não foram votados como focal? Por quê? Há baixa frequência tarefa de baixo valor que foram focal? papéis focais muitas vezes executar tarefas focais. Isso é verdade de seus modelos? Detalhes interessantes surgem de discutir estes pontos.

Passo contextos de interacção 8. Extrair

Um contexto de interação é um lugar no software aonde um utilizador pode ir para executar algumas das suas tarefas. É o trabalho do designer para encontrar esses lugares, organizá-los de forma adequada, e em seguida, colocar as ferramentas certas lá para a execução da tarefa ocorre sem problemas.

Por exemplo, se eu estivesse projetando para um projeto chamado "minha casa", eu poderia ter algumas tarefas chamados de "arrumar mantimentos", "fazer o jantar", e "ouvir rádio." Eu poderia decidir essas três tarefas pertencem juntos em um contexto interação eu vou chamar de "cozinha". Em minha cozinha eu vou colocar ferramentas que preciso para executar as tarefas: uma geladeira, um fogão, panelas e frigideiras, e um rádio. Você vai encontrar contextos de interação no software que você usa todos os dias.

Você verá que os clusters em seu modelo de tarefas são susceptíveis de ser contextos de interação. As tarefas agrupados porque eles eram semelhantes, por algum motivo; muitas vezes, porque eles são realizados em momentos similares por funções de utilizador semelhantes.

Nome cada contexto interação e escrever o nome em um cartão 3x5. Organizar estes cartões em outra folha de papel cartaz. Corrigi-los com fita adesiva quando o arranjo se sente bem. Desenhar linhas entre cada cartão contexto de interação para indicar como um utilizador pode navegar a partir de um contexto para outro. Isto é um *mapa de navegação.*

Neste ponto, você deve começar a ser capaz de visualizar o software. Um contexto de interação bem nomeado torna-se um nome de módulo adequado e uma boa maneira para se referir às tarefas realizadas lá (assim como quando eu digo "cozinha" que você pode imaginar rapidamente os tipos de tarefas que eu poderia fazer lá). Você pode achar que é útil para as seções de índice do plano de projeto

por contexto de interacção. É mais fácil explicar que você está trabalhando sobre as funções de "processamento ordem" do que está a relacionar todas as tarefas individualmente.

Passo 9. Identificar um plano de lançamento incremental

Você tem na frente de você um conjunto de folhas cartaz que mostram os papéis que irão utilizar o sistema e as funções do sistema precisa para fornecê-los. O que você ainda precisa descobrir é a ordem em que para entregá-los para a comunidade de utilizadores. Você pode estar fazendo várias entregas, caso em que você deseja entregar as funções em um *útil*
ordem. Se você está indo só para fazer uma única entrega, então você ainda quer desenvolver e integrar as funções na mesma ordem, para proteção contra a pior situação caso, que você executar fora de tempo antes de chegar a tudo.

Meu [Jeff escrevendo aqui] ordem preferida é "mais alto valor de negócio composta." Eu uso essa frase porque, às vezes, para entregar um fio tarefa completa, end-to-end, você realmente tem que desenvolver e integrar um ou mais tarefas que têm menor valor individual. Se você não incluir essas tarefas, em seguida, os utilizadores não vai realmente obter o valor integral das tarefas mais valiosos (focais). Queremos agregar valor ao negócio completo, o valor composto de todo o encadeamento de tarefas ao longo de um fluxo de trabalho.

O caminho mais curto, mais útil é o que eu chamo de "span sistema 27. "Desenvolver este primeiro, como um *Victory início* para os utilizadores e que eles vão considerar o *caminhando de esqueleto* da funcionalidade do sistema. liberações subseqüentes incrementalmente adicionar recursos para essa extensão do sistema.

Para construir o plano de lançamento incremental, fazer uma cópia dos cartões de tarefas. Você pode entregar copiar todos os cartões de tarefa. Acho que introduzir-los em uma planilha, em seguida, imprimi-las em cartões usando um processador de texto funciona bem. Eu faço isso durante uma pausa ou no almoço nesse mesmo dia.

Coloque cada cartão de tarefa em uma folha (fresco) Cartaz de acordo com o seu tempo e criticidade.

- **Tempo corre a partir da esquerda para a direita, referindo-se quando uma função de utilizador pode executar essa tarefa particular.**

- **criticidade de negócio funciona superior (mais importante) para baixo (menos importante). Criticidade se refere a como necessária a tarefa é executar (tarefas opcionais nunca será na linha superior.)**

Você vai encontrar a tarefa cartões de começar a organizar-se em fileiras. A linha superior conterá tarefas críticas dispostos em ordem de tempo. Cada linha subseqüente irá conter outras tarefas, menos críticos, também em ordem de tempo. Desenhar uma linha diretamente sob a linha superior. Este é o tempo de sistema, o menor conjunto de recursos que podem ser liberados para executar um fluxo de processos de negócio end-to-end.

27 Emprestado de magra Desenvolvimento de Software (Poppendieck 2003).

Finalmente, é a vez dos desenvolvedores. Têm os desenvolvedores apresentam dar estimativas de tempo de desenvolvimento ásperas em cada cartão tarefa 28. estimativas curso de grãos são muito bem neste momento. Costumo restringir as estimativas para uma semana no mínimo, 3 ou 4 no máximo. Desenhe linhas esquerda para a direita através do modelo aonde você gostaria de tarefas de grupos para versões futuras. O que você tem aqui não é o cronograma do projeto, mas uma maneira de identificar grupos de tamanho similar de função para desenvolver em cada iteração, informações úteis para a construção do *Mapa projeto*.

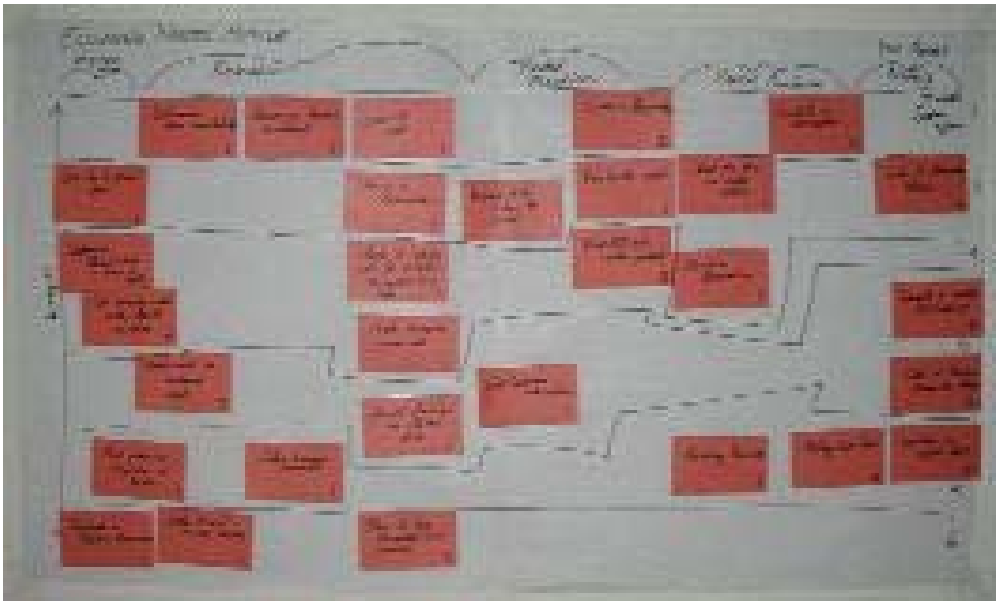


Figura 3-14. Um plano de extensão, que permite ver tanto o fluxo de trabalho e possíveis lançamentos de relance. (Graças a Jeff Patton)

Isto conclui a sessão, o trabalho cansativo, mas divertido que fortalece as relações entre os utilizadores, patrocinadores e desenvolvedores.

Certifique-se de colocar os cartazes com destaque na área de desenvolvimento como radiadores de informação, constantemente lembrando os desenvolvedores dos recursos do produto, os papéis focais e tarefas focais, e as conversas em torno de como eles chegaram a ser. Isso permite que os desenvolvedores para ver o software através dos olhos daqueles papéis de utilizador, em vez de através dos olhos de um gerente de projeto, desenvolvedor ou testador.

* * *

Derivando a UI

28 Notas de Alistair: Em primeiro lugar, estes devem ser vistos como estimativas de dimensionamento relativos, não decorrido estimativas de tempo.

Pode ser prejudicial se comprometer com um cronograma de desenvolvimento tão rapidamente. Eu iria verificar novamente esses números mais tarde usando *Planificação Blitz* ou uma técnica semelhante. Você é provável encontrar tarefas adicionais a serem feitas que não foram notados durante a corrida deste workshop.

Quando se trata tempo para desenvolver o protótipo de interface do utilizador, considere uma técnica descrita por Constantino, Windl, Noble, e Lockwood 29, que eu [Jeff] também praticar e ter um pouco modificado para atender meu estilo pessoal.

Faça o seguinte para cada tarefa do utilizador, em folhas cartaz:

- **Escrever em duas colunas uma sinopse do diálogo entre o utilizador eo sistema. Coloque o *Intenções de utilizadores* na coluna da esquerda e o correspondente *Responsabilidades do sistema* Na direita. Isto é o que Constantino chama de um "caso de uso essencial" (Constantine 1999 ???).**

- **Coloque notas pegajosas junto às intenções do utilizador e as responsabilidades do sistema, que serão elementos de interface do utilizador. Coloque aqueles para áreas de entrada e ações do utilizador no lado esquerdo e aqueles para a recipientes de informação no lado direito.**

  Por exemplo, uma intenção do utilizador, como "utilizador identifica auto" sugere duas áreas de entrada e uma acção (nome, senha, algum tipo de botão "Go"). Escrever "username" e "password" em duas notas. Escrever "ação" em um terceiro, ou simplesmente usar uma com um tamanho ou cor diferente para indicar sua presença. Furar todos os três ao lado da frase "utilizador identifica auto."

  Para uma frase como "sistema exibe pedidos em aberto," você quer um componente mostrando uma lista de pedidos. Escrever "lista de pedidos" em uma nota pegajosa e colocá-lo ao lado da frase "sistema exibe pedidos em aberto."

  Quando estiver pronto, o caso de uso deve ser manchado com notas pegajosas.

- **Transferir as notas para outro flipchart, representando telas o utilizador verá, colocando-os em áreas que parecem apropriados para a interface de utilizador do sistema. Quando você estiver confortável com a colocação, substituir cada nota pegajosa com um desenho simples do componente desenhado aproximadamente do tamanho que você esperaria o componente para aparecer na interface do utilizador. Faça isso com cada componente. Agora você tem o que é chamado de "wire-frame" desenho da interface do utilizador. Testar a interface de utilizador do fio-frame, referindo-se de volta para o caso de uso. Um participante desempenha um papel utilizador que executa a tarefa, o outro desempenha o papel do sistema. Certifique-se a interface de utilizador que você projetou suporta fácil desempenho da tarefa.**

* * *

Inspeção de Usabilidade (durante o projeto)

É tempo para inspecionar seu projeto (em execução).

Projetar o software rodando em uma parede suficientemente grande para um grupo de participantes para vê-lo facilmente.

Escolha um participante para realizar a função de utilizador que estará usando a parte do software que está sendo inspecionado. Prefiro alguém não familiarizado com a funcionalidade. Dar a essa pessoa uma tarefa ou lista de tarefas a serem executadas, escolhido a partir do modelo de tarefas que você desenhou no início do projeto ou a partir dos casos de uso essenciais.

---

29 http://www.foruse.com/articles/canonical.htm

A pessoa que executa a tarefa explica em voz alta o que está fazendo. Como as informações do sistema exibe explicar você pode explicar o que estão vendo. Todos na sala observa defeitos ou alterações sugeridas em cartões de índice.

Após as tarefas foram realizadas no sistema, recolher os cartões. Eliminar duplicatas, esclarecer questões, e priorizar as questões a serem abordadas pela equipas de desenvolvimento. (Não ser deprimido se você encontrar dezenas de problemas em sua primeira passagem, isso é normal. Apenas certifique-se resolver os problemas.)

Repetir este processo conforme necessário até que as interacções do utilizador durante a execução de tarefas de mover-se suavemente.

* * *

QA Testing as Personalidades do sistema

Finalmente, você terá que confirmar que o software acabado é apropriado para a função de utilizador que vai usá-lo. Muitas vezes temos uma pessoa QA fingir ser um utilizador na parte de um papel indicado pelo modelo.

Para cada função no modelo de papel, assuma o conhecimento e objetivos desse papel. Use a descrição do papel e qualquer outra informação que você pode reunir sobre as pessoas que podem entrar nesse papel. habilidades método de atuação vir a calhar aqui.

Executar todas as tarefas atribuídas a esse papel usando o sistema em execução, e escrever para baixo, como se você fosse uma pessoa nesse papel, quaisquer problemas que você sente a pessoa terá.

Você pode achar que você executar a mesma tarefa várias vezes. Cada vez, porém, você vai estar fazendo isso da perspectiva de uma função de utilizador diferente, e avaliá-lo de forma diferente.

* * *

Jeff resume:

> **design centrado no utilizador utiliza informações sobre as características e objetivos dos utilizadores em todo o processo de desenvolvimento e, finalmente, valida o sistema contra essa informação. Descobri isso geralmente resulta em mais fácil de software de utilizador que os utilizadores finais reais são mais felizes com. Descobri que geralmente têm muito menos retrabalho sobre os recursos acabados e descobrimos menos recursos imprevistos tarde no desenvolvimento.**

Técnica 7. **Miniature processo**

Qualquer novo processo é estranho e desconcertante. Quanto maior a duração do processo, quanto mais tempo antes de novos membros da equipas compreender como as várias partes do ajuste processo com o outro. Você pode acelerar esse entendimento, reduzindo o tempo gasto pelo processo, usando o *Miniature processo.* Aqui estão três exemplo de seu uso:

> **Uma grande organização, utilizando uma metodologia muito maior do que claro, colocar cada novo funcionário através de um projeto de uma semana. Até o final desse projeto, o novo funcionário tinha exercido todas as partes do processo. Ele ou ela poderia, então, trabalhar em um projeto real saber a conexão entre todas as atividades de processos e produtos de trabalho.**

> **"Hora Extreme" foi inventado por Peter Merel para introduzir as pessoas para o XP em 60 minutos 30. O grupo é executado duas iterações de meia hora: 10 minutos para o jogo de planeamento, 15 minutos para desenvolver uma solução de aceitação e testes, cinco minutos para o teste de aceitação. O grupo atravessa o processo duas vezes para que eles possam experimentar a redução do escopo do projeto a meio de uma iteração, empurrando características de uma iteração para a próxima iteração, e adicionar e requisitos mudando dentro e entre iterações. Para fazer com que o tempo de desenvolvimento de trabalho em 15 minutos, a equipas projeta algo arbitrária, como uma ratoeira ou um dispositivo de pesca, e só desenha seu projeto em transparências.**

> **Para uma pequena empresa, 50 pessoas com 16 programadores, uma vez eu corri a 90 minutos *Miniature processo* que é necessário programação, criação de testes, checkin-checkout e integração entre as três camadas de arquitetura. Foram utilizadas duas iterações de 45 minutos e um problema de programação muito, muito simples: um contador de cima para baixo executado através de uma interface web. Levou duas tentativas para passar o problema em 90 minutos. A primeira tentativa foi utilizado como uma sessão de prática para entender o que estava sendo solicitado. A segunda tentativa foi feita ao vivo na frente de toda a empresa como uma demonstração do método de trabalho.**

Você pode, com a introdução de Crystal Clear usando um *Miniature processo* algures entre 90 minutos e um dia. Isso pode ser feito antes do workshop inicial metodologia de modelagem, ou logo após, então a equipas pode "provar" a sua nova metodologia. O workshop metodologia de modelagem em si pode ser recolhidos por meio de executá-lo de uma forma muito digerido durando apenas meia hora, apenas para que a equipas pode aprender como ele funciona em um tema "mais seguro" do que seu projeto.

---

30 http://c2.com/cgi/wiki?ExtremeHour

Muitas técnicas podem ser introduzidos usando um *Miniature processo* para reduzir o problema que as pessoas têm com começando a usar um processo desconhecido. Você pode executar o seu primeiro

***workshop de reflexão* em apenas 15 minutos como uma *Miniature processo.* Eu corro um *Miniature processo* para escrever casos de uso no meu curso de caso de uso para que as pessoas possam ver o quadro geral antes de entrar em detalhes.**

Técnica 8. **Side-by-Side Programação**

"Programação em pares" envolve duas pessoas trabalhando em uma atribuição de programação, em uma única estação de trabalho 31. Algumas pessoas acham que isso seja demasiado união. Side-by-side de programação permite-lhes obter alguns dos efeitos da programação em pares sem abandonar sua indivíduo atribuições (outros programação ou).

Na programação lado-a-lado, duas pessoas sentar-se perto o suficiente para ver telas uns dos outros com facilidade, mas trabalhar em suas próprias atribuições. Esta é uma amplificação de Comunicação osmótica para o contexto de programação.

Na foto abaixo, Justin e Andrew definiu as estações de trabalho para que os monitores são apenas dois pés afastados. A ideia é que Justin só deve ter que virar a cabeça ou inclinar-se um pouco para ver a tela de Andrew. Então, eles podem trabalhar em suas respectivas atribuições, mas cada um pode pedir a outra a qualquer momento para olhar para um pedaço de código, executar um teste, ou similar.

Figura 3-15. Side-by-side de programação. (Graças a Jeff, Justin, Andrew e Tomax)

Estudos têm demonstrado que de decoração e revisões de código são formas rentáveis para reduzir defeitos no software (McConnell ref ???). As pessoas acham que é difícil encontrar a energia e interesse para qualquer convocar uma reunião de código-revisão ou assistir a um. Sendo ao lado uns dos outros, no entanto, essas duas pessoas podem examinar

e comentar sobre pequenas quantidades de código em curtos períodos de tempo, com pouca cerimônia ou interrupção de seu trabalho. Ele fornece a "peer peering código" descrito por Cristal no primeiro dos e-mails no início do livro.

Os desenvolvedores que primeiro me disse desse método disse que lhes permitiu trabalhar em paralelo em sua programação, para que eles não foram tanto ocupada pela mesma tarefa. Ao mesmo tempo, eles poderiam rever o código do outro, tal como solicitado pela complexidade do código. Eles estavam programando um sistema cliente-servidor no momento, um trabalho sobre o cliente, a outra no servidor. Pode-se dizer para o outro: "Eu tenho o esboço compilado. Você pode testá-lo?" A outra seria executado o sistema, e, em seguida, retornar à sua programação enquanto a primeira pessoa fixo qualquer que seja bug apareceu no teste.

31 Vejo *Pair Programming Iluminado (* Williams, 2002).

Outro desenvolvedor sênior comentou que ele passou a maior parte do dia escrevendo relatórios e gestão de conversas, em vez de programação. Tendo seu sit apenas dois passos de distância de outro programador significava que ele poderia ser útil como treinador e segundo par de olhos, sem ter que desistir de seu trabalho não-programação.

Jim Coplien especulou que em tempo integral par programação não é a quantidade ideal de tempo para as pessoas para passar juntos 32. Side-by-side de programação permite que as pessoas de misturar e combinar o tempo que passam na mesma tarefa e em tarefas separadas.

Tal como acontece com muitas das ideias em Crystal Clear, você pode encontrar side-by-side de programação para ser um trampolim útil para fazer Extreme Programming, ou como uma queda-back no caso de pessoas não levam à prática completa XP.

32 Comunicação pessoal.

Técnica 9. **queime gráficos**

queime gráficos **tornaram-se uma maneira favorita para dar visibilidade sobre o progresso do projeto. Eles são extremamente simples e surpreendentemente poderosa. Eles revelam a estratégia a ser utilizada, mostrar os progressos realizados contra previsões, e abrir a porta a discussões sobre a melhor forma de proceder, incluindo as discussões difíceis sobre se a cortar escopo ou estender o cronograma. Eles têm um mapeamento natural para as cartas de valor agregado utilizados em projetos militares / governamentais. Eles devem parte de seu saco padrão de truques para o planificação e elaboração de relatórios do projeto.**

Acontece que a embalagem de uma casa tem semelhanças marcantes para o desenvolvimento de software, tanto quanto o planificação e ir rastreamento. Eu uso o seguinte exercício para ilustrar o uso de cartas de queimaduras:

> **Imagine que você tem 30 dias para arrumar a casa que você está vivendo em (com um casal de filhos, apenas para torná-lo mais doloroso). Suponha que há 14 quartos (ou equivalentes quarto) em 3 andares. Traçar o seu plano para que você sempre sabe o que é feito, o que resta, e quão bem você está fazendo.**

A maioria das pessoas planeja para embalar a casa como mostrado na Figura 3-16. Eles planejam primeiro jogar fora todo o lixo, então embalar os itens não-críticos, e embalar os itens de vida essenciais nos últimos dias. Este é, naturalmente, a técnica que eu usei as duas primeiras vezes eu estava nessa situação, e eu posso falar com a dor em primeira mão ao dizer que é terrível.

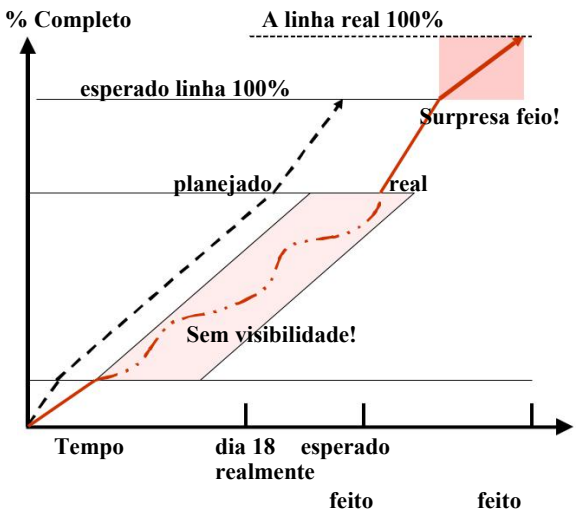


Figura 3-16. Agendamento sem marcos definitivos.

A grande falha nela é a ausência de marcos intermediários claros.

É praticamente impossível dizer se um é de 60% ou 70% feito (falando com software e sala de embalagem). Para a maioria do projeto, vê nem o verdadeiro tamanho da tarefa em mãos, nem a taxa (mais lento do que o esperado) do progresso. Faltando tanto daqueles, é impossível dizer quando um vai ficar feito. Em ambos os casos, uma surpresa feio espera no final, quando uma miríade de pequenos itens inesperados de repente se tornam visíveis (pensar sobre quando você carregado todas as caixas em um caminhão em movimento e voltou para a casa, apenas para descobrir todos os tipos de coisas que, de repente "apareceu" quando tudo o resto foi retirado). O resultado clássico em

projetos de software é que os programadores continuam relatando que as coisas são 80% completo, ou "Vamos ser feito quando estamos a fazer."

A reparação é encontrar "interessantes" e claros marcos intermediários. Na embalagem, esses marcos são quartos completamente lotado e vazios, contendo "nem mesmo uma meia" (e possivelmente com fita da polícia do outro lado da porta de entrada para manter as pessoas de dumping coisas de volta para eles!). O pensamento é que se não é tanto como um meia esquerda, então não é, possivelmente, outra coisa além do meia, eo relato é falho. Veremos uma aplicação desta ideia "meia" em apenas algumas páginas.

Em software, esses marcos são a entrega de execução, o código testados, tal como descrito em Entrega frequente.

Esta nova estratégia tem implicações que tornam a vida um pouco estranho (novamente, tanto para software e casa de embalagem). Na casa de embalagem, as pessoas têm de sair de seus quartos, e, eventualmente, a cozinha se torna inutilizável. Em software, os testes têm que ser executado re- e projetos e documentação devem ser alteradas em cada entrega. Estes não são livres - custo e desconforto ambos vêm com a estratégia.

O retorno para o custo eo desconforto é maior visibilidade, melhor acompanhamento, e menos probabilidade de superação projeto inesperado. Na maioria dos casos, esse benefício vale bem a pena o custo.

Vamos olhar para esta estratégia em um projeto de software.

- **Faça uma lista de todos os itens a serem entregues. Você pode trabalhar a partir do *Projeto Mapa, Blitz Planificação* ou jogo de planificação do XP. Associar com cada item de entrega um *custo relativo.* Tentar associar com cada item também um parente *benefício do negócio,* para que você possa discutir com o *Patrocinador executivo* o valor a ser emitido para o negócio ao longo do tempo (Figura 3-17 mostra um fragmento de uma tal lista). A razão para usar estimativas relativos em vez de absolutos é que se você for, por exemplo, 10% ao longo do seu primeiro conjunto de itens, todos os itens restantes serão dimensionados de acordo. Os gráficos de queimadura irá mostrar isso automaticamente.**

- **Ordenar e sequenciar os itens de trabalho por dependência de desenvolvimento, custo e valor, e agrupá-las em uma seqüência, como mostrado na Figura 3-18 (você pode reconhecer a semelhança do que estamos fazendo aqui eo que acontece em um *Planificação Blitz* sessão).**

- **Estimar o quanto pode ser feito em cada período de iteração ou entrega. Desenhar uma linha sob o último item que se encaixa em cada um. Numerar os lançamentos. Eu vim para chamar isso de *Lista iceberg.* A parte "acima da água" lista todos os itens que podem ser entregues no ciclo de entrega atual. Os "abaixo água" listas de peças cada, que serão entregues em ciclos de entrega posterior. O nome reflete que quando você adiciona um novo item à parte de água acima, ele empurra tudo para baixo, e algo que estava acima da água cai abaixo da água (não será entregue no ciclo de entrega atual). o**

***Lista iceberg* é particularmente útil em projetos em que os requisitos mudam frequentemente.**

Na terminologia scrum, a parte de água abaixo é referido o *Product Backlog* e a parte de cima da água é a *Sprint Backlong* 33.

A principal diferença entre esta lista e a estrutura de trabalho-quebra engenharia de sistemas padrão é que *você não obter créditos para qualquer item que não resulta em execução, testado código.* OK, você também obter crédito para *resultados finais* tais como materiais de treinamento e documentação de entrega.

33 Mais sobre atrasos do Scrum está disponível online em http://wiki.scrums.org/index.cgi

Característica	recurso Descrição	Valor.	dias Est ideais	Husa. dias decorridos
1 Interface de utilizador			8	
1.1 Configuration gera comentários do item de linha ordem	Permitir a configuração para preencher comentários item de linha ordem com informações detalhadas de configuração, juntamente com comentários de configuração. mudança parada de item de linha observa que não seja de dentro do configurador apropriado.	M	2	5
1.2 Configrator UI Retrabalho: estilo assistente Verbose	Retrabalho de interface do utilizador para coincidir com o prototye discutido na sessão de revisão colaborativa. Rápido, entrada de dados simples. Sem frescura desnecessários.	H	6	15
1.3 Configuração gera preços específicos do cliente	Configuração é passada ao cliente ordem atual para uso na determinação de preços específicos de clientes.	H	5	12,5
1,4 permitir a adição de uma configuração e continuar	Actualmente um cego configurado pode ser adicionado a um pedido. Para adicionar um cego semelhante posterior que você precisa para voltar para configurador e reinserir todas as informações. Esta característica iria permitir ao utilizador para aceitar uma configuração acabada imediatamente e permitir a adição de uma configuração adicional, usando os dados da configuração anterior.	eu	2	5
2 Especial Order PO Geração			5	
2.1 PO gerado correctamente para a configuração	Atualmente POs são geradas para itens de pedidos regulares. Mudar para permitir que os componentes de configuração para gerar 1 ou mais ordens de compra a um custo fornecidos pelo componente.	H	5	12,5
2.2 Criar / confirmar itens de fornecedores existir para skus	confirmar que um item do fornecedor existe para cada sku utilizado para tratamentos de janela de base e opções. Aonde existe nenhum item de fornecedor para um fornecedor atribuído, criar dinamicamente o item do fornecedor.	H	1.5	3,75
2,3 geração PO é automática	Atualmente Pos pode ser gerada após uma ordem especial é salvo e um depósito é feito. Permitir a geração PO automática immideately após o depósito for ofertadas, ou immideately se não é necessário depósito.	eu	4	10
3 Order avançada			9	
3.1 Order Formulário avançado mostra mais detalhes	Alterar ordem avançada forma para mostrar um "overflow" são para cada item de linha, aonde informações adicionais, tais como comentários, número de Po, PO data do pedido etc .. Pode ser mostrado.	M	4	10

Figura 3-17. Extrato da lista de recursos com estimativas de tamanho e valor de negócio (Graças a Jeff Patton)

Nome do recurso	Módulo	Valor	Dev Raw. Tempo (dias ideais)	Dias decorridos estimado	lançamento #
1.1	Configuração gera item de linha de pedido comenta	M	2	5	1
1,2	Configrator UI Retrabalho: estilo assistente Verbose	H	6	15	1
2.1	PO gerado correctamente para a configuração	H	5	12,5	1
2.2	Criar / confirmar itens de fornecedores existir para skus	H	1.5	3,75	1
3.1	Encomendar Formulário avançado mostra mais detalhes	M	4	10	1
3.2	Ordem cumprida pelo custo PO	H	2	5	1
3,3	pedidos de repetição trabalha com configurações cegos	M	3	7,5	1
3,4	comentários de configuração são visíveis, não editável	eu	1	2,5	1
4.1	Base de Dados de mudança hierarquia	H	1.5	3,75	1
4,2	Estilo pode localizar gráficos de preços com base na seleção de cores	H	2	5	1
4.3	Atribuir cores específicas do grupo de cor para Preço gráficos	H	2	5	1
4,4	mensagens separadas de opções	H	2	5	1
4,5	questões separadas de opções	H	2	5	1
1.3	Configuração gera preços específicos do cliente	H	5	12,5	2
1,4	Permitir a adição de uma configuração e continuar	eu	2	5	2
2,3	geração PO é automática	eu	4	10	2
3,5	Order mostra quadrados. Filmagem e rodando comprimento	eu	2	5	2
3,6	Vendedor itens que podem ser solicitadas mostrar detalhes adicionais	eu	2	5	2
3.7	tratamentos de janela impressos mostram renúncias	M	6	15	2
3.8	Permitir a cópia fim item de linha	H	2	5	2
4.1	estilo associado com Retail.net DCL	eu	1	2,5	2
4,11	grelha	eu	1	2,5	2
4,6	Atribuir cliente específico descontos de venda	H	3	7,5	2
4.7	lookup do cliente	H	1	2,5	2
4.8	Tipo de conta de cliente pesquisa	M	1	2,5	2
4,9	Opções pode ter exigido perguntas	eu	3	7,5	2

Figura 3-18. o *Lista iceberg* com *Product Backlog. (* Graças a Jeff Patton)

- **Neste ponto, você pode fazer uma *Queime* gráfico (Figura 3-19). Marcar tanto unidades de trabalho relativos ou% completas no eixo vertical, e tempo de calendário na horizontal. Desenhar uma linha desde a origem até o ponto de conclusão, quer por estimar a taxa em que o trabalho pode ser realizado, ou como muitas vezes acontece, o "cair morto" prazo de entrega. Isso mostra o progresso "planejado".**

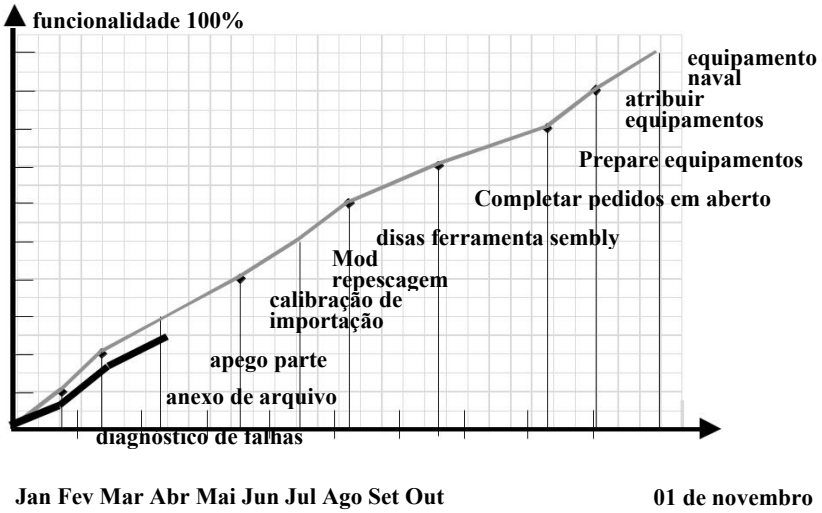


Figura 3-19. o *Queime* gráfico mostra rapidamente o progresso e valor agregado. A linha fina longo mostra a prevista funcionalidade e programação, a linha grossa mostra o progresso curto até à data. Após cada iteração, marcar a quantidade de trabalho em relação ficou concluída. Você vai descobrir que com apenas duas marcas no gráfico que você pode ver como muito mais lento do que você está se movendo do que o esperado! Sim, isso pode ser uma má notícia, mas pelo menos é *más notícias detectado precocemente,* ao contrário de *más notícias detectado tarde demais para fazer qualquer coisa sobre ele!*

O gráfico de queima-se assemelha-se o valor de ganho gráfico clássica (Lett 98, Fleming 88). A diferença essencial é só isso pessoas incluem tradicionalmente na carta-valor agregado *tarefas concluídas,* ou não resultou em código ficando integrado. Em projetos ágeis, o crédito é dado somente quando o código é integrado e testado (a regra "não sobra meia"). A estratégia ágil produz informações mais confiáveis sobre o estado real do projeto do que o geral.

o *Queime* gráfico tem muitas características interessantes.

- **Ele mostra tanto o estado e taxa de progresso ("velocidade") de uma forma que é clara e fácil de discutir.**

- **Se você parcela o valor do negócio no eixo vertical em vez de esforço de desenvolvimento, você tem um gráfico que mostra corretamente o quanto o valor de ter entregue ao longo do tempo. Plotagem ambos no mesmo gráfico pode ajudar a lembrar a equipas sobre aonde o valor de negócio real encontra-se em seu trabalho. Se você sequenciar o *Lista iceberg* flutuar as maiores itens de valor de negócio para o topo, e acompanhar o valor de negócio entregue, então não importa quando o projeto fica encerrado, será claro para todos que eles têm o maior valor de negócio no tempo gasto.**

- **Finalmente, a carta-burn-se é uma parte fundamental de uma boa de uma página relatório status do projeto (exemplos de relatórios de status são mostrados no capítulo do Word Produtos). Tais folhas de status página um- simplificar o trabalho dos executivos olhando para os relatórios de muitos projetos.**

Mesmo que a carta-burn-se é muito bom, há uma surpresa desagradável se você não for cuidadoso na escolha das unidades que você usa para o eixo vertical.

Na casa de embalagem, a unidade óbvia de estimativa e monitoramento é a caixa de embalagem. Na programação, a unidade óbvia é a linha de código (LOC). Estes são convenientes e dar detalhe maravilhoso.

O problema é que você não vai saber o número real de caixas ou linhas de código necessário até o fim (tons de "Nós vamos ser feito quando estamos a fazer."). As faixas gráfico *taxa* maravilhosamente, mas *progresso* mal. Não só você não vai descobrir má notícia cedo, mas você não vai saber quantas caixas ou LOC você precisa até o último item está embalado ou programado. Que seja tarde demais.

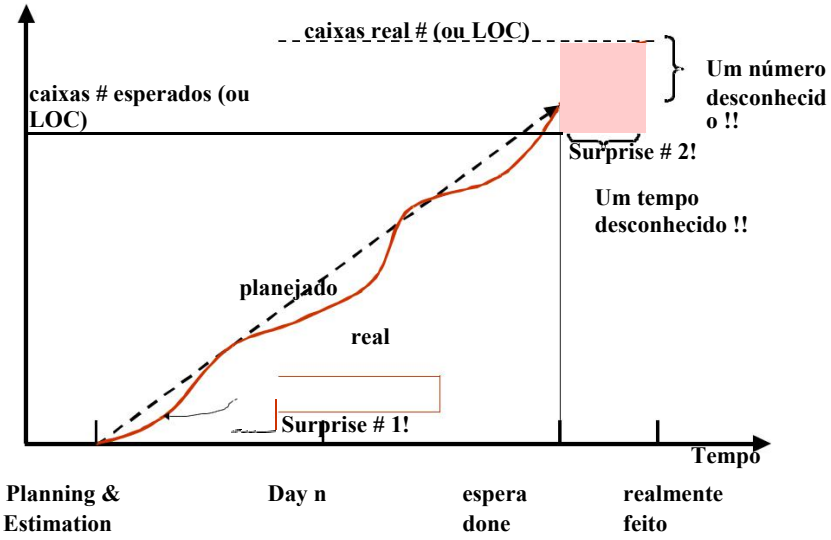


Figura 3-20. UMA *Queime* traçar quando as unidades erradas são medidos.

A reparação é escolher uma unidade de medida que não pode expandir. Na casa de embalagem, a medida de substituição é fácil: usar as salas em vez de caixas (que seria uma grande surpresa para descobrir um novo quarto depois de ter embalado tudo o resto). A resposta geralmente não é óbvia em um projeto de software. Você pode ter sorte e ter um número relativamente estável de casos de uso, ou um número fixo de módulos para substituir, mas não há resposta de aplicação geral.

Conheci duas equipas que tinham encontrado uma resposta para si. Eles estavam fazendo projetos de substituição do sistema aonde os subsistemas comunicadas em um padrão de fluxo de trabalho. Sua unidade não-expansão foi o componente de fluxo de trabalho. Essas equipas foram capazes de colorir cada subsistema amarelo como eles começaram a trabalhar sobre ela, e depois verde, uma vez que foi concluída. Eles foram capazes de usar "componentes substituídos" como sua unidade não-expansão (ver a folha de estado da amostra no capítulo Produtos de Trabalho).

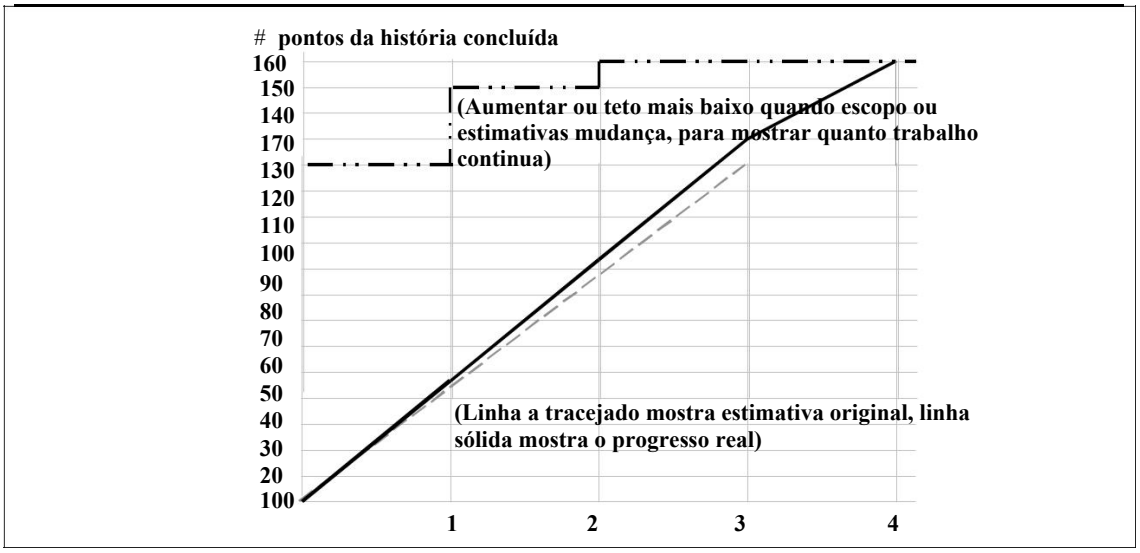


Figura 3-21. UMA *Queime* Gráfico que mostra previsto e realizações reais, e mudando linha de 100%, utilizando o estilo Goodwin-Rufer.

Na situação mais usual, o escopo do projeto faz em mudança fato de ao longo do tempo. Phil Goodwin e Russ Rufer introduziu a idéia de marcar o levantamento e abaixamento do teto, ou marca de 100%, no decorrer do projeto. Esse mapeamento permite que a equipas para mostrar o plano original e realizações reais, mesmo em face de mudanças no escopo. A figura abaixo mostra o gráfico estilo Goodwin-Rufer queimar-se em uma situação aonde a equipas realmente teve um desempenho melhor do que as suas estimativas, mas teve de entregar âmbito aumentando. (O projeto realmente usou o gráfico burn-down Scrum padrão mostrado na Figura 3-24. Eu acho que você vai concordar que suas conversas com os patrocinadores teria sido mais fácil se eles tivessem usado tanto Figura 3-21 ou 3-26).

Se você tiver sorte o suficiente para ter uma boa unidade não-expansão de medida, então você pode usar o *Queimar* gráfico, que algumas pessoas acham emocionalmente mais poderoso.

o *queimar* gráfico é emocionalmente poderoso, porque há um sentimento especial sobre bater o número zero que ajuda as pessoas a ficar animado sobre completar o seu trabalho e pressionando para a frente. Eu posso falar em primeira mão ao dizer que, assim como nós mudamos nosso packing house gráfico para usar a salas como a unidade de medida e colocar os quartos em um *queimar* gráfico, nossa confiança de que nós realmente cumprir a nossa agenda aumentaram dramaticamente. Eu vi o mesmo se aplica a uma equipas de projeto de software.

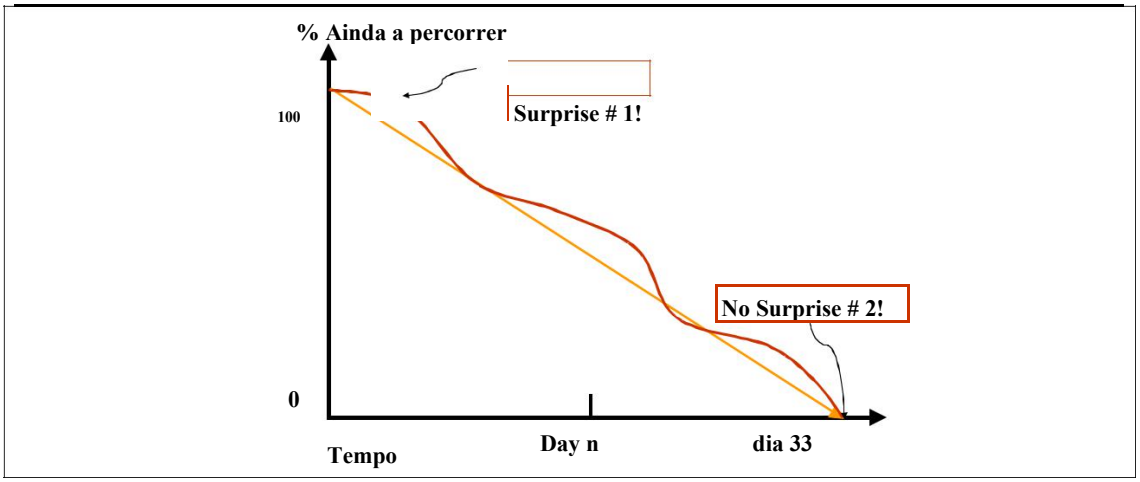


Figura 3-22. o *Queimar* gráfico.

É **fácil dizer: "Nós não podemos usar o gráfico burn-down, porque nada é fixo em um projeto de software." Estranhamente, os primeiros a dizer isso foram aqueles que trabalham no projeto de substituição do sistema que acabamos de mencionar. Após um pouco de análise, descobrimos que eles tinham exatamente o problema-packing house na frente deles. Eles foram capazes de converter imediatamente para o gráfico burn-down, utilizando subsistemas substituído como sua unidade não-expansão de medida.**

* * *

Há três rugas deixadas para discutir.

A primeira é a pequena mentira Eu apenas disse no último parágrafo sobre unidade não-expansão do trabalho. A equipas descobriu depois de um curto período de tempo que a sua *Patrocinador executivo* manteve acrescentando exigências de novas funcionalidades, tanto no sistema antigo e a substituição. Em outras palavras, apesar de terem sido prometido que o alcance seria fixo (uma vez que o tempo eo orçamento era), que não era verdade. Como se isso não bastasse, ele estava realmente oposição ao projeto em si, e foi obrigado a apoiá-lo só por causa da pressão da comunidade de utilizadores (veja como utilizador poderoso apoio da comunidade pode ser!).

Isso colocou o time em uma posição maldito-se-você-do-e condenado-se-você-não. Eles não podiam relatar que um subsistema estava completa (nem mesmo uma meia esquerda nele) se ele estava indo para continuar jogando mais meias de volta para o quarto, e eles não poderia impedi-lo de jogar mais meias.

Sua eventual estratégia tem alguma influência sobre gráficos de queimaduras, mas também alguma influência sobre o papel positivo dos gerentes intermediários. O gerente intermediário decidiram que iriam levar a sua missão projeto como inicialmente previsto, para substituir o sistema existente. Eles iriam entregar esse sucesso, e acompanhar todas maior margem como a crescente actividade "pós-substituição".

Eles mapearam este como mostrado na Figura 3-23. Com esta estratégia e gráfico, eles foram capazes de manter a sua Foco sobre a questão crítica de substituir o sistema existente, mostrar que eles estavam monitorando as novas funções sendo adicionadas, e manter o *Patrocinador executivo* de descarrilar seu projeto.

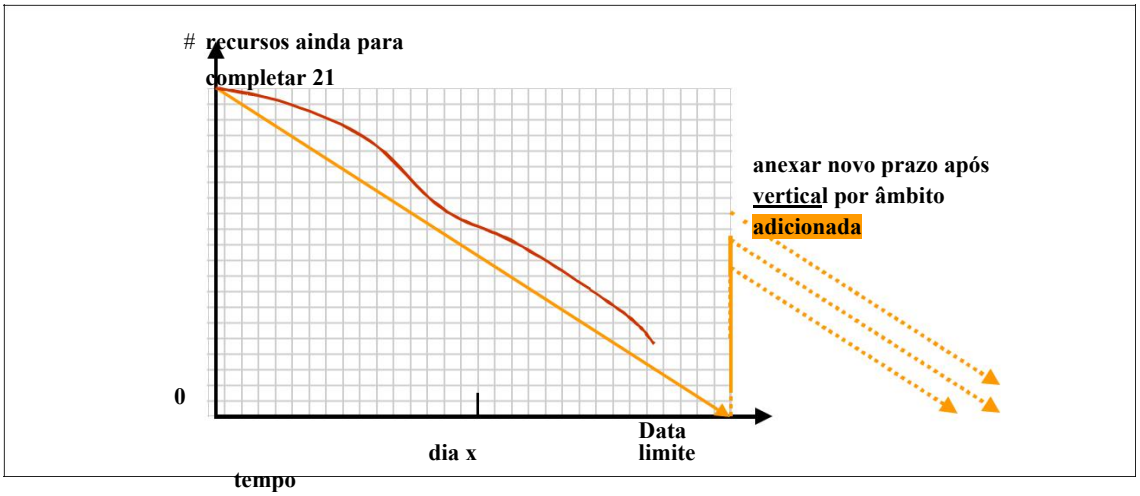


Figura 3-23. o *Queimar* gráfico para um projeto de disco-prazo com aumento âmbito adicionado no final.

Mike Cohn relata uma situação semelhante em seu livro *User Stories,* embora em circunstâncias amigáveis. A equipas recebeu 130 pontos da história (unidade relativa de tamanho para as histórias de utilizador XP) para entregar em três iterações. Após cada iteração, os patrocinadores adicionou novas histórias e a equipas revisou os tamanhos relativos dos andares restantes. Ambos aumentou a estimativa da quantidade de trabalho restante. Mike e eu discutimos como eles devem traçar o progresso da equipas contra a mudança escopo, assumindo que eles gostam do poder emocional do gráfico burn-down.

A Figura 3-24 mostra o que a equipas mostrou os patrocinadores em suas reuniões (Cohn 2003). Mike e eu senti que este quadro esconde informação muito importante. A Figura 3-25 mostra como eu pensado para modificar o gráfico para ilustrar as realizações efectivas contra o âmbito da mudança, e a Figura 3-26 mostra como Mike proposta para modificar o gráfico para mostrar a situação de mudança. Você pode notar que 3-26 é a versão para baixo do gráfico queimar-up Goodwin-Rufer.

Oferecemos estes exemplos de gráficos para mostrar maneiras diferentes em que você pode mostrar previstos e realizações reais com simplicidade e poder expressivo. Há provavelmente mais variações para descobrir, mas estes devem ajudar a começar.

Figura 3-24. Scrum-style *queimar* gráfico como mostrado para os patrocinadores do projeto (após Cohn 2004).

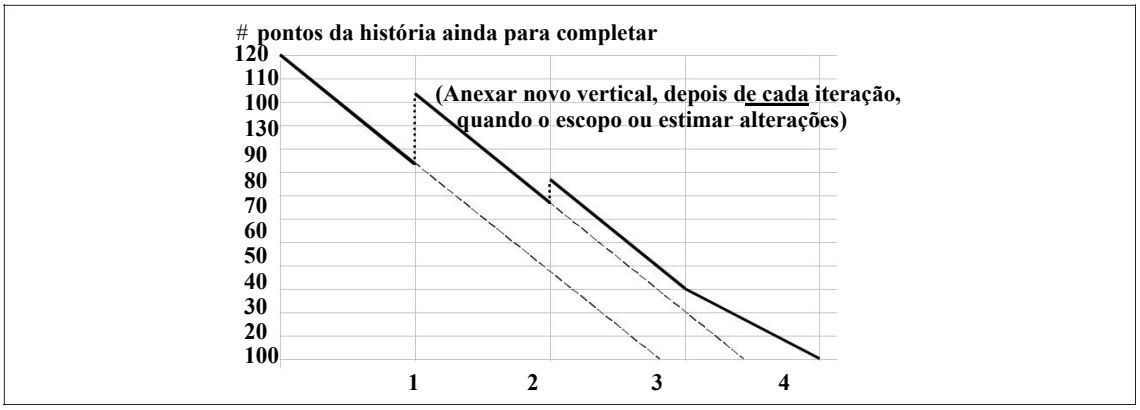


Figura 3-25. *Queimar* carta para o mesmo projeto mostrando aumento âmbito após cada iteração.

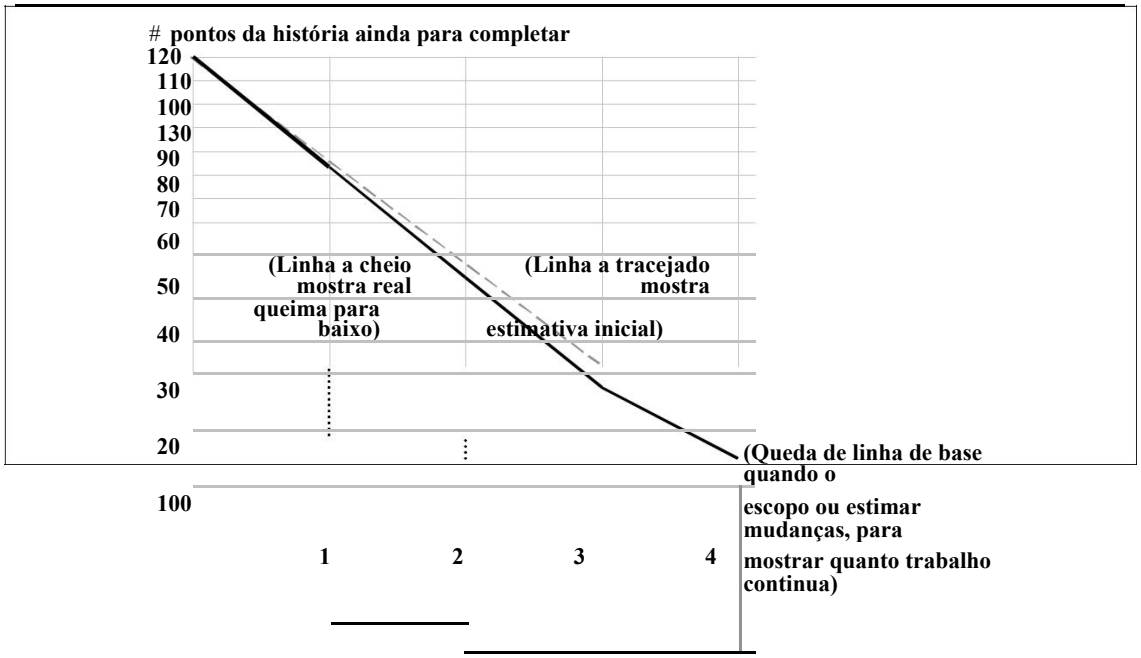


Figura 3-26. *Queimar* gráfico para o mesmo projeto usando a versão 'down' do gráfico Goodwin-Rufer.

O segundo enrugamento a ser discutido é o tamanho da unidade seleccionada de medida. Isto é muito relevante no início de um projeto porque a equipas irá quase sempre se movem mais lentamente no início do que o esperado, e apenas *quão* muito mais lentamente é uma peça valiosa de informações.

Vamos voltar para o packing house exemplo para ver isso claramente. Os gráficos na Figura 3-27 mostram que mais grossa será a unidade de medida, quanto mais longo o período de bloqueio, durante o qual não há informação disponível. Tendo "caixas embaladas" já rejeitaram como uma boa unidade de medida, podemos escolher "quartos embalados" ou "pisos embalados", como unidades não-expansão.

A Figura 3-27 mostra *queimar* gráficos para ambas as opções. Em ambos os casos, as pessoas ficam para trás na mesma velocidade. No entanto, com piso como unidades, eles não descobrir o quão ruim a notícia é até um terço do caminho através do projeto, que é um tempo relativamente longo. Usando quartos como unidades, eles descobrem o quão ruim a notícia é depois de apenas alguns dias. A moral da história é escolher a unidade não expansível menor de medida.

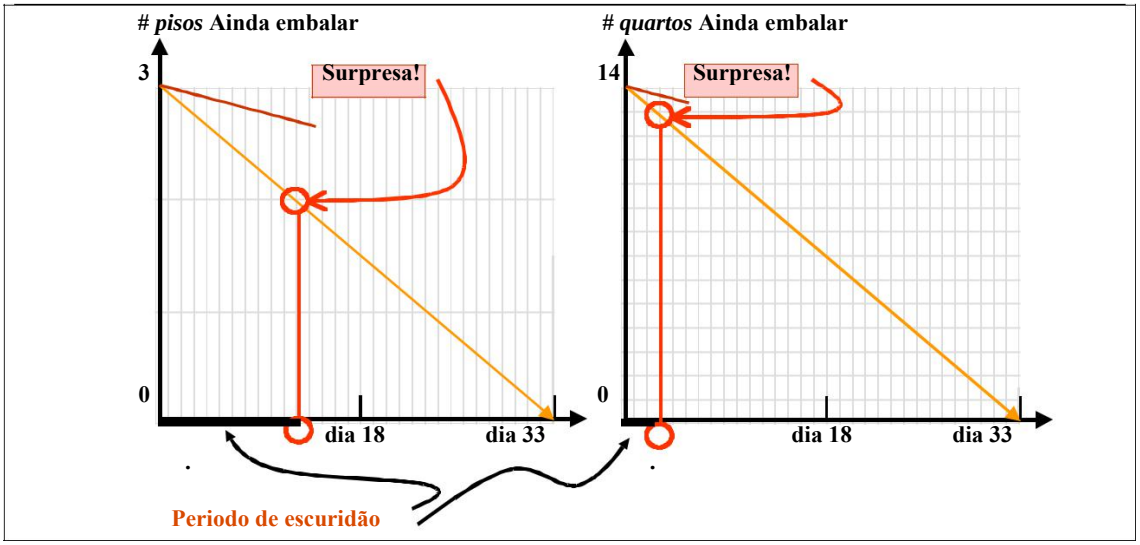


Figura 3-27. *Queimar* paradas por duas unidades de base diferentes. Quanto menor for a unidade, mais curto será o período negro-out inicial.

* * *

O item final para discutir está a lidar com itens que mudam as prioridades. Isso nos leva de volta para o *Lista iceberg,* que tem demonstrado grande utilidade em projetos em que as prioridades mudam frequentemente e violentamente.

> **Um grande projeto foi a entrega de valor a um cliente em uma base regular. O cliente iria mudar as atribuições de trabalho e prioridades tantas vezes e tão tarde que nenhum plano projeto foi significativa. O cliente foi, no entanto, satisfeito com o software a ser entregue, assim que mantiveram solicitando mais trabalho feito.**
>
> **Em desespero, o gerente de projeto basta colocar todos os recursos solicitados em uma única lista priorizada com tamanhos estimados. Ele anunciou que tudo o que foi acima da "linha de água", como calculado a partir das estimativas de tamanho e velocidade da equipas, que são entregues no próximo lançamento, e tudo o mais seria entregue "mais tarde". O cliente pode inserir, remover, modificar ou priorizar recursos a qualquer momento, mesmo que afeta a versão atual. Sem recorrer ao argumento, todas as partes poderia calcular o que chegaria a entrega durante este ciclo eo que foi empurrado para trás.**

O nome *Lista iceberg* vem desde o nascente e afundando efeito a lista mostra: a adição de itens de alta prioridade para o topo da lista faz com que alguns itens de água acima de afundar abaixo da linha de água; removendo itens acima água pode provocar um novo aumento item acima da linha de água). Assim como com um iceberg, a porção de água acima é apenas uma fracção do tamanho total.

o *Lista iceberg* é útil em ambientes hostis, aonde os patrocinadores mudam as exigências e prioridades em uma base diária e sem aviso. Em tal

ambiente, postar o atual *Lista iceberg* em um lugar altamente visível, e certifique-se o conteúdo dos próximos

lançamentos são *derivada* em vez de argumentar.

	Nome do recurso Módulo	Valor	Dev Raw. Tempo (dias ideais)	Revisto estimativa dias ideal	liberação #
1.1	Configuração gera item de linha de pedido comenta	M	2	2.6	1
1,2	Configrator UI Retrabalho: estilo assistente Verbose	H	6	7,8	1
2.1	PO gerado correctamente para a configuração	H	5	6,5	1
2.2	Criar / confirmar itens de fornecedores existir para skus	H	1.5	2	1
2,3	geração PO é automática	H	4	5.2	2
3.1	Encomendar Formulário avançado mostra mais detalhes	M	4	5.2	1
3.2	Ordem cumprida pelo custo PO	H	2	2.6	1
3,3	pedidos de repetição trabalha com configurações cegos	M	3	3,9	1
3,4	comentários de configuração são visíveis, não editável	eu	1	1.3	1
4.1	Base de Dados de mudança hierarquia	H	1.5	2	1
4,2	Estilo pode localizar gráficos de preços com base na seleção de cores	H	2	2.6	1
4.3	Atribuir cores específicas do grupo de cor para Preço gráficos	H	2	2.6	1
4,4	mensagens separadas de opções	H	2	2.6	1
4,5	questões separadas de opções	H	2	2.6	1
1.3	Configuração gera preços específicos do cliente	H	5	6,5	2
1,4	Permitir a adição de uma configuração e continuar	eu	2	2.6	2
3,5	Order mostra quadrados. Filmagem e rodando comprimento	eu	2	2.6	2
3,6	Vendedor itens que podem ser solicitadas mostrar detalhes adicionais	eu	2	2.6	2
3.7	tratamentos de janela impressos mostram renúncias	M	6	7,8	2
3.8	Permitir a cópia fim item de linha	H	2	2.6	2
4.1	estilo associado com Retail.net DCL	eu	1	1.3	2
4,11	grelha	eu	1	1.3	2
4,6	Atribuir cliente específico descontos de venda	H	3	3,9	2
4.7	lookup do cliente	H	1	1.3	2
4.8	Tipo de conta de cliente pesquisa	M	1	1.3	2
4,9	Opções pode ter exigido perguntas	eu	3	3,9	2

Figura 3-28. o *Lista iceberg* a partir da Figura 3-18 revisto sobre a segunda iteração. A Figura 3-28 mostra como isto funciona, a partir do exemplo em Figure3-18. As estimativas do tamanho do mudaram, e "geração PO automatizado" foi movida para cima da linha de água. Como resultado, "mensagens separadas / perguntas de opções" caiu abaixo da linha de água e foi adiada para uma versão posterior.

Reflexão sobre as estratégias e técnicas

Crystal Clear afirma explicitamente que nenhuma estratégia ou técnica é obrigatória. Por que eu descrever estes queridos e não outros?

Ambos moldar a metodologia e Melhoria reflexivo são elementos de Crystal Clear necessário, mesmo que nenhuma técnica específica é obrigatória para eles. A maioria das pessoas nunca estas coisas, e preciso ver uma amostra para começar.

A técnica de oficina de reflexão particular, eu descrevo é curto, eficiente e permite que as pessoas tanto expressar seus pensamentos e, em seguida, voltar ao trabalho. Se você estiver fazendo qualquer tipo de retrospectiva mensal que permite a livre discussão, então você deve ser capaz de experimentar diferentes estilos de workshops em meses sucessivos, e gerar o seu próprio estilo pessoal dentro de alguns meses.

eu descrevo *Planificação Blitz* como uma alternativa para o XP "jogo de planificação." As duas diferem de três maneiras:

- **A lista de cartões de jogo de planificação *histórias de utilizadores,* e a *Planificação Blitz* lista de cartões *tarefas;***
- **O jogo de planificação tem as pessoas assumem não há dependências entre as histórias, enquanto *Planificação***

  ***Blitz* tem as pessoas a analisar as dependências entre as tarefas;**

- **O jogo de planificação assume iterações de comprimento fixo, enquanto *Planificação Blitz* não assume nada sobre duração da iteração. Há momentos em que essas diferenças são de interesse. *Planificação Blitz* permite que a**

equipas

- **ver a forma do trabalho à frente deles (produzindo o *Projeto Mapa),***
- **trabalhar fora, aonde o *caminhando de esqueleto* deve aparecer eo que ela pode consistir de,**
- **trabalhar fora o menor conjunto de tarefas antes de receita pode começar a fluir, e**
- **relatar aos patrocinadores o número de tarefas internas que precisam ser executadas antes de funcionalidade útil se torna visível.**

Estas diferenças são de particular importância no início do projeto. Mais tarde no projeto, a equipas pode ser capaz de simplesmente listar os recursos a serem desenvolvidos e os custos de desenvolvimento relativo.

o *Miniature processo* permite que as pessoas a construir um pequeno filme interna de como um novo processo funciona. Desde que se mudou para Crystal Clear é susceptível de envolver um monte de mudança nos hábitos de trabalho, é útil reduz a tensão e incerteza.

o *Exploratórios 360 °* dá às pessoas uma chance de pegar certos tipos de erros fatais, erros que muito possivelmente todos (incorretamente) assume alguém já verificado. Desde o *Exploratórios 360 °* não leva muito tempo, tem um alto valor de retorno para o tempo necessário. Ele também alinha o time na missão e abordagem do projeto.

***Victory cedo, Walking Skeleton* e *rearquitetura incremental* se encaixam. Mesmo desenvolvedores muito experientes discordam sobre o quanto a arquitetura a desenvolver no início do projeto. Estas três estratégias descrever uma maneira de tirar proveito da incrementais**

desenvolvimento para estabelecer segurança e valor nos estágios iniciais do projeto, e ainda lidar com os erros e aprendizagem que formam uma parte natural do desenvolvimento arquitetônico.

Radiadores de informação **são úteis em cada ponto no projeto. A maioria das equipas ágeis fazer grande uso deles, mas a maioria das equipas fora da comunidade ágil não aproveitar o suficiente destes dispositivos. Deve ser uma estratégia pronunciado usar toda a parede disponível para captura de informações de utilidade para a equipas do projeto e os seus visitantes (existe o perigo de sobrecarregar o espaço visual da parede, mas eu só encontrei uma equipas que tinha chegado a esse ponto). Uma vez que você começar a usar *Radiadores de informação* você pode achar que você deseja reorganizar o escritório para fazer mais paredes disponível.**

o *Diário stand-up* reunião foi introduzida na metodologia Scrum. Tenho ouvido de pessoas adicionando esta técnica única para cada tipo de projeto e metodologia para um bom efeito, e por isso estou feliz para espalhar seu uso.

incluo *Interaction Design Essencial* porque há tão pouca literatura sobre como trabalhar de design de interface do utilizador e design de interação em um projeto ágil. Esta adaptação do design centrado no uso se encaixa em ambos os projetos XP e cristalina. Você pode querer estudar mais dos escritos de Jeff Patton e no livro de Lucy Lockwood e Larry Constantine para se adaptar e aperfeiçoar o uso do mesmo.

queime gráficos **são extremamente úteis na fase de planificação do projeto, porque eles mostram a carga de trabalho de forma tão clara e as pessoas podem responder com comentários sobre a viabilidade do plano. Eles também são adequados como relatórios de status, pela mesma razão: clareza imediata.**

Crystal Clear permite que uma equipas de usar as técnicas que funcionam melhor para seus membros. Prefiro esperar que simplesmente como o curso normal da actividade profissional, Crystal Clear praticantes vai aprender cada uma das técnicas e estratégias listadas neste capítulo, e descobrir muitos mais como eles vão.

Capítulo 4 Explored (O processo)

Crystal Clear utiliza processos cíclicos aninhados de vários comprimentos: o episódio desenvolvimento, a iteração, o prazo de entrega, eo projeto completo. O que as pessoas fazem a qualquer momento depende de aonde eles estão em cada um dos ciclos. Este capítulo lineariza os ciclos na medida do possível, e aponta algumas das suas interações.

A maioria dos processos de desenvolvimento descritas de 1970 a 2000 foram descritas como sequências de passos, mesmo quando o processo de iterações e incrementos recomendado. Esta confusão sempre causada entre os leitores. O texto apareceu para ditar um processo de "cachoeira", enquanto os autores mantiveram afirmando que não era.

Estas descrições de processos de aspecto linear sofre de dois problemas. A primeira é que eles realmente descrever o gráfico de dependência entre os produtos de trabalho, *fluxo de trabalho.* Um gráfico de dependência devem ser lidas em ordem back-to-front: A equipas não pode entregar o sistema até que seja integrado e corre. Eles não podem integrar e testar o código até que ele é escrito e em execução; Eles não podem desenhar e escrever o código até que eles aprendem quais são os requisitos (e assim por diante). O gráfico de fluxo de trabalho não exige um processo em cascata, mas simplesmente reflete dependências.

Quando um processo é descrito pelo gráfico de dependência de fluxo de trabalho, as pessoas mentalmente tratar a imagem de gráfico de dependência como incentivo para completar cada produto de trabalho mencionado antes de iniciar o produto de trabalho seguinte no gráfico. Isso raramente é uma boa estratégia no desenvolvimento de software. Ao olhar para um gráfico de dependência de fluxo de trabalho, as questões relevantes a fazer são estas:

- ***Quantos* requisitos, *em que nível de conclusão,* são necessários antes que o design pode utilmente começar? (As respostas são "relativamente poucos" e "relativamente baixo", respectivamente).**

- ***Como grande parte do sistema* deve ser concebido e programado antes da integração útil e teste pode ser feito? (A resposta é "relativamente pouco.")**
- ***Como grande parte do sistema, em que nível de correção* é necessário antes de ser útil para os utilizadores ou adequado para revisão do utilizador? (As respostas são "relativamente pouco" e "relativamente baixo", respectivamente).**

Os baixos níveis de cada realmente necessário vem como uma surpresa para muitas pessoas.

***desenvolvimento simultâneo* é o nome dado quando o trabalho prossegue em paralelo em produtos de trabalho dependentes. Em desenvolvimento simultâneo, muitas ou todas as atividades estão em jogo ao mesmo tempo. Pessoas trocar notas continuamente para manter a par da mudança de estado de cada parte. desenvolvimento simultâneo não é novo, seja para o desenvolvimento de software ou gestão de projetos em geral 34. É feito por quase todos os equipas do projeto I visitar, embora poucos tenham notado eles fazem isso.**

desenvolvimento concomitante está descrito em pormenor no *Agile Software Development,* no *Corrida do ouro* estratégia *Sobrevivendo Projetos Orientados a Objetos,* e geralmente é pressuposta na maioria dos meus escritos. Portanto, não vou tentar descrever desenvolvimento simultâneo aqui, nem o gráfico de dependência em Crystal Clear (que deve ser bastante óbvio). A remoção desses dois tópicos permite-me para descrever outros aspectos interessantes do processo.

---

34 *Desenvolvimento simultânea (???* 1997) contém uma excelente descrição sobreposta fases em projetos de engenharia civil.

O segundo problema que flagela descrições lineares do processo é que os processos estão a tentar descrever são cíclicos, utilizando vários ciclos de diferentes comprimentos 35.

Percebo sete ciclos em jogo na maioria dos projetos:

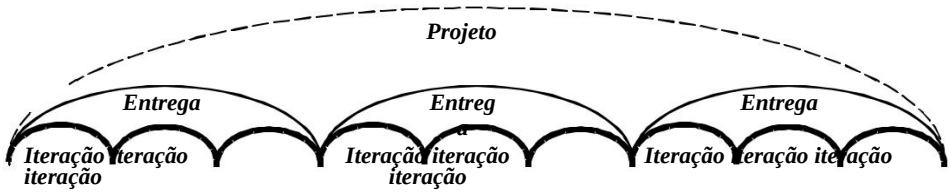
- **O projeto (uma unidade de financiamento, o que poderia ser qualquer duração)**
- **O ciclo de entrega (a unidade de entrega, de uma semana a três meses)**
- **A iteração (uma unidade de avaliação, desenvolvimento e celebração, uma semana a três meses)**
- **A semana de trabalho**
- **O período de integração (uma unidade de desenvolvimento, integração e testes do sistema, 30 minutos a três dias)**
- **O dia de trabalho**
- **O desenvolvimento *episódio (* desenvolvimento e verificação em uma seção do código, tendo alguns minutos a algumas horas)**

Crystal Clear requer várias entregas por projeto, mas não múltiplas iterações por entrega. Vou repetir este ponto várias vezes, mas gostaria de chamar sua atenção para ele imediatamente.

As equipas que usam períodos de iteração curtas dentro de um ciclo de entrega longo, por vezes, adicionar um "super ciclo" de talvez quatro ou seis iterações, para fornecer um ritmo intermediário. Eles usam esse ciclo super para refletir sobre seu processo, para comemorar, para relaxar. Não vou elaborar muito sobre o super-ciclo neste capítulo. As equipas podem derivar que elementos dos ciclos de iteração e entrega para passar para o ciclo de super.

Cada ciclo tem a sua própria sequência, o seu próprio ritmo. Em um determinado dia, diferentes atividades serão em jogo a partir dos vários ciclos. As atividades mudam de hora a hora, dia a dia e semana a semana. Isso faz uma descrição linear completo do processo virtualmente impossível.

As figuras seguintes mostram várias maneiras de desenrolar destes ciclos. Estes esboços simples tem que omitir algumas das interações entre os ciclos, alguns dos quais eu vou apontar em breve. Observe que episódios, dias e períodos de integração pode aninhar dentro de uma iteração de várias maneiras. As Figuras 4-2 e 4-3 mostram duas maneiras possíveis. Há outras combinações válidas ainda mais difícil de desenhar.



35 Sou grato a deserição Don Wells' do XP como ciclos aninhados. Ver http: // www. extremeprogramming.org/map/loops.html

Figura 4-1. ciclos de iteração e entrega dentro de um projeto.

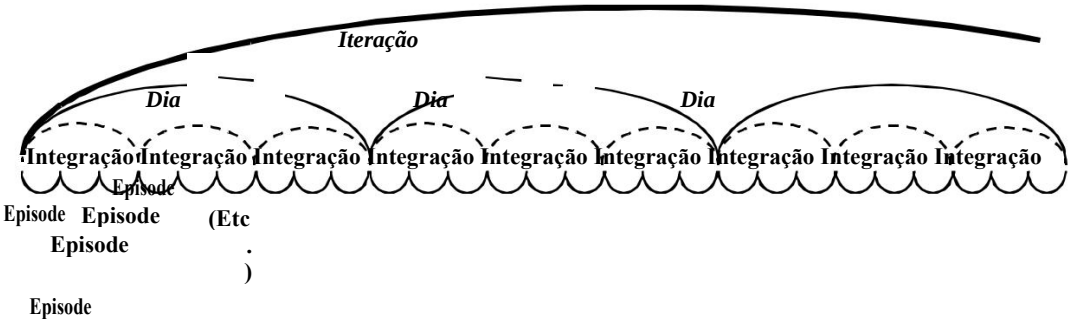


Figura 4-2. Um ciclo de iteração, integrando várias vezes por dia.

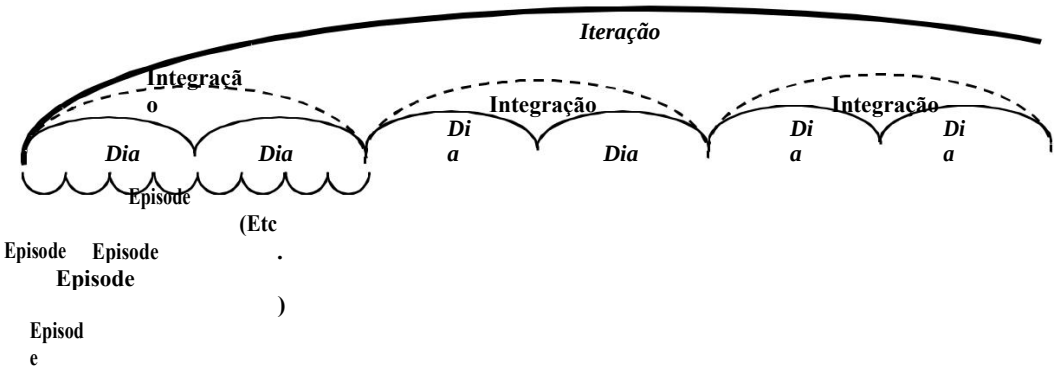


Figura 4-3. Um ciclo de iteração com vários dias por integração. A Figura 4/4 mostra a expansão de cada ciclo separadamente, listando as actividades específicas que ocorrem especificamente para o tipo de ciclo. A Figura 4-5 mostra estas actividades em uma sequência de tempo plausível (verticalmente para baixo a página), colocando cada actividade em uma coluna para o seu tipo de ciclo.

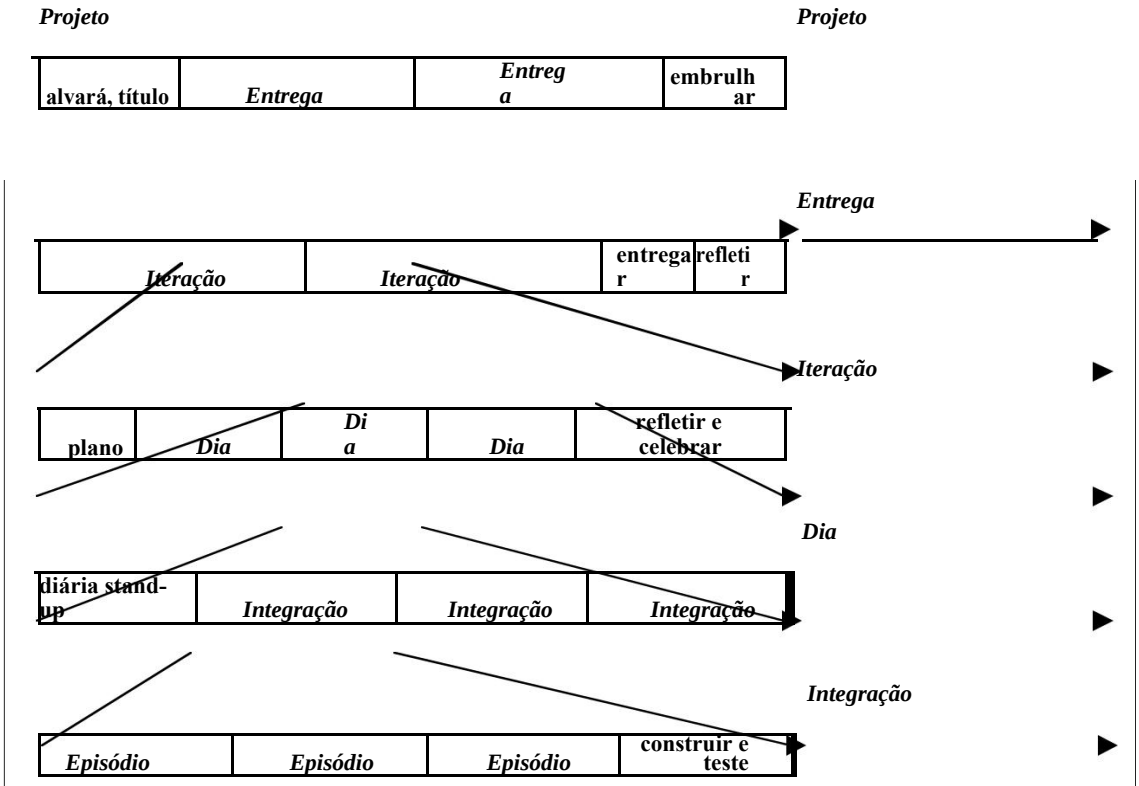


Figura 4-4. Os ciclos expandida para mostrar atividades específicas.

Projeto Iteração Dia integração Episode
Alvará, título
Plano
standup diário
Design e check-in Design & Entrada
Construir e testar Design e check-in
Design e check-in
Construir e testar
standup diário
Design e check-in Design & Entrada
Construir e testar Design e check-in
Design e check-in
Construir e testar
entregar
Reflectir e celebrar Plano
(etc.)
Embrulhar

Figura 4-5. As actividades de uma sequência plausível verticalmente.

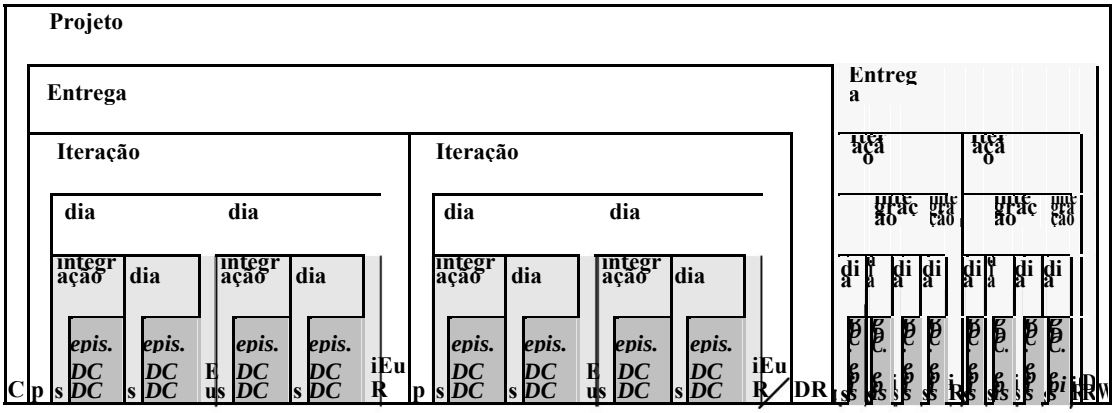


Figura 4-6. Os ciclos desenrolou todo o caminho para mostrar as atividades diárias. Finalmente, na Figura 4-6, que têm desenrolado todos os ciclos e sombreado os ciclos de maneiras diferentes, de modo a actividade pode ser ligado ao seu ciclo. As letras maiúsculas e minúsculas representam projeto C Hartering, iteração p Lanning, diariamente s tandup, d esenvolvimento,

c Parreira-in, Eu NTEGRAÇÃO, R oficina eflection, D elivery, e projeto W Rap-Up. Claro, eu tinha que escolher uma relação particular entre o ciclo diário e o ciclo de integração para este desenrolar particular.

As pessoas interessadas em detalhe vai notar que há uma linha traçada através do R no final da última iteração de um ciclo de entrega. Isso indica que a equipas não "refletem, entregar, refletir de novo", mas sim, mais naturalmente espera até depois da entrega efectiva, em seguida, reflete apenas uma vez.

Outro detalhe é que a sessão de planificação na primeira iteração de um ciclo de entrega é susceptível de ter duas partes: o ajuste do plano de projeto de granulação grossa e criar um plano de iteração de granulação fina 36.

Estes são os tipos de pequenos ajustes de processo que a maioria das equipas fazem naturalmente, mas fazer processo cíclico tão difícil descrever de forma explícita.

As seções a seguir expandir em todos, mas o ciclo semanal.

36 Obrigado, Jeff Patton, para tornar isso explícito para mim.

O Ciclo do Projeto

Um projeto como um ciclo? Embora cada projeto financiado como uma vez por e para-toda a actividade, ele é normalmente seguido por um outro projeto, num ciclo que se repete. É útil para às equipas de desenvolvimento de organização e de se habituar a esse ciclo.

Um ciclo de projeto em Crystal Clear tem três partes:

- uma *fretamento* atividade,
- uma série de dois ou mais ciclos de entrega,
- um ritual completo: o wrapup projeto.

fretamento

o *fretamento* atividade leva alguns dias a algumas semanas. Ele consiste em quatro etapas:

1. Construir o núcleo da equipas,
2. Executar o *Exploratórios 360 ° (* o que pode resultar no cancelamento do projeto),
3. Forma e afinar as convenções metodologia,
4. Construir o plano de projeto inicial.

Note-se que ter apenas um ciclo de entrega é uma violação da Crystal Clear (por alguma definição da palavra "entrega"). Um projeto grande o suficiente para precisar de uma descrição metodologia irá beneficiar de vários ciclos de entrega. Já vi projetos que o patrocinador pensei que era pequeno o suficiente para não precisar de várias entregas que fizeram muito mal pela simples falta de feedback que vem no final de uma entrega. Nos raros outros casos, o projeto foi tão pequeno que é um caso de "apenas fazê-lo e ir para casa." Se não é uma questão de "apenas fazê-lo e ir para

casa," por favor mandar para pelo menos dois ciclos de entrega. Verifique as secções livro sobre Entrega Frequent e Ciclo de entrega para saber o que constitui uma entrega adequada.

Fretamento: construir a equipa

Um projeto do tipo que vai usar Crystal Clear muitas vezes começa com um *Patrocinador executivo* e uma *Designer-chefe,* e, eventualmente, um utilizador chave.

Normalmente, dois a cinco outras pessoas são adicionados ao projeto, com vários mix de habilidades, experiência e habilidade. Os mais novatos na equipas, o mais difícil é para o designer-chefe de trem e desenvolver software. Portanto, se há mais de quatro pessoas envolvidas, a equipas deve incluir pelo menos um outro desenvolvedor experiente e competente, fazer backup e descarregar a liderança. Deve haver uma pessoa experiente e competente para cada duas ou três pessoas júnior ou novatos.

O patrocinador executivo

o *Patrocinador executivo* é a pessoa que em última análise, importa que o projeto é feito. Esta pessoa fornece o dinheiro - ou pelo menos arranja para ele aparecer - e fornece orientação essencial para o grupo. Esta pessoa destaca as prioridades do grupo precisa estar ciente, e recebe o projeto não apenas apoio monetário, mas também apoio logístico e emocional. Muitas vezes, o *Patrocinador executivo* é também o especialista de domínio.

Durante o projeto, a *Patrocinador executivo* precisa para lembrar a equipas de desenvolvimento do seu verdadeiro objetivo e mantê-los informada de quaisquer mudanças na direção de negócios que afeta a direção do projeto. Em troca, a equipas precisa manter o *Patrocinador executivo* avaliado do andamento do projeto (queimar gráficos são excelentes para isso).

Um perigo está à espreita para o projeto com um altamente técnico *Patrocinador executivo:* sobreviragem. Muitas vezes acontece, especialmente para pequenos projetos, que o projeto de *Patrocinador executivo* Era uma vez um designer técnica soberba. Este tipo de pessoa não pode resistir soletrando para a equipas de design apenas como eles *poderia* implementar o sistema, *Não que eles têm de,* claro. Em alguns casos, essa pessoa é correta, mas ainda fica no caminho da equipas. O ex-top designer que está patrocinando um projeto terão que aceitar que a equipas não pode criar um design tão excelente como ele ou ela poderia, e viver com isso.

Uma das maneiras de manter o ex-top designer de interferir muito no projeto é a utilização de casos de uso corretamente escritos para as especificações do sistema (ver *Casos de Uso eficaz da escrita).* casos de uso escrito corretamente especificar com precisão o comportamento do sistema desejado sem especificar o design.

Eu tenho que mencionar que, por vezes, a *Patrocinador executivo* faz, na verdade, saber o que é possível com a tecnologia, ou o que é necessário, mesmo que isso parece difícil. Nestes casos, a melhor situação é para a equipas patrocinador e design para desenvolver uma forma de discussão que permite que o patrocinador para expressar o que ele ou ela sabe sem correr sobre a equipas. Afinal, uma das contribuições primárias do *Patrocinador executivo* é a motivação da equipas.

O Designer de chumbo

A pessoa dada a atribuição de *Designer-chefe* tem uma descrição do trabalho complicado, que normalmente não é escrito com antecedência. Esta pessoa é muitas vezes o melhor designer na equipas, uma pessoa que poderia, em princípio, projetar todo o sistema. Esta pessoa também é suposto para treinar os membros mais jovens da equipas, falar com o *Patrocinador executivo* e os utilizadores finais, construir um plano de projeto e gerenciar o desenvolvimento do projeto e ao mesmo tempo projetar a parte mais difícil do sistema. É uma descrição do trabalho impossível, exceto que ela é realizada com sucesso por muitos (excesso de trabalho) pessoas ao redor do mundo.

A melhor coisa a *Patrocinador executivo* pode fazer para o *Designer-chefe* é fornecer um segundo desenvolvedor especialista. Esta segunda desenvolvedor especialista deve ser o único que lida com as tarefas de design mais difíceis, porque esta segunda desenvolvedor especialista não é tão provável de ser pego em reuniões sobre os utilizadores, planos, status, interfaces e assim por diante.

Um perigo chave em torno do *Designer-chefe* é que essa pessoa pode ficar queimado antes que o projeto é longo. Em um caso, eu conheci um *Designer-chefe* que já estava trabalhando 60 horas por semana no início do projeto. Seu próprio plano de projeto tinha-lhe ensinando Java para os novos programadores, bem como fazer as tarefas de design mais difíceis, bem como. . . bem, você começa a idéia. Embora comum, esta situação não é saudável. No momento em que o projeto passa a marca de metade dessa pessoa é susceptível de estar a trabalhar 90 horas por semana e dormindo sob a mesa por algumas horas cada noite. Algum tempo depois disso, uma calamidade de algum tipo é susceptível de embater no projeto, e ainda é necessário mais tempo com essa pessoa.

Foi exatamente por esse motivo que Jens Coldewey concebeu a estratégia de redução de riscos de gestão de projetos que se referem como Capacidade Líder Spare 37. Essa estratégia aproximadamente diz: "Assegurar a liderança técnica tem alguma capacidade disponível para ajudar em caso de emergência." Sabemos que situações de emergência surgem na maioria dos projetos; se o líder técnico não tem capacidade de reposição, ele ou ela não pode trabalhar através de emergência. Começando um projeto de 60 horas por semana não deixar o líder técnico com muita capacidade ociosa, e em 90 horas por semana, que a capacidade de reposição é tudo ido.

> No caso do *Designer-chefe* Acabei de mencionar, nós trabalhamos em conjunto para identificar todas as tarefas que alguém poderia fazer, mesmo que ele era o mais adequado para o próprio tarefa. Ao tomar essas cargas fora dele (ele ainda teria que configurar ou monitorá-los), fomos capazes de reduzir sua carga de trabalho para cerca de 45-50 horas por semana no início do projeto, para que ele possa ser capaz de manter dentro de cerca de 60 horas durante o período de alta carga de projeto.

O Utilizador Embaixador 38

O sistema será mais adequado para o que os utilizadores precisam se os desenvolvedores têm acesso a um especialista sobre o uso do sistema. O valor do sistema traz para a organização é muito sensível às informações os desenvolvedores a obter sobre o contexto de uso. Isso significa que é importante que a equipas tem fácil acesso a especialistas uso adequado.

Como muito do tempo em que o utilizador faz a equipas precisa?

Seria bom ter o utilizador especialista disponíveis para a equipas o tempo todo, é claro. Isso raramente é prático. Algumas metodologias sugerir ou exigir o especialista utilizador estar presente 50% do tempo. Que também seria maravilhoso, é claro, mas também é geralmente impraticável.

A minha experiência é que, se a equipas pode telefonar para o *Embaixador Utilizador* periodicamente durante a semana para fazer perguntas, e recebe algumas horas (até mesmo um preço tão baixo quanto duas horas) de tempo dedicado a cada semana, eles podem trabalhar através de suas perguntas, obter uma boa informação requisitos, mostra demos e obter uso e projeto do gabarito de um bem tipo adequado.

---

37 XP usa o termo "ritmo sustentável." Enquanto que é som, acho estratégia Jens' para ser útil mais específico.

38 I pedir o termo da metodologia DSDM (DSDM Consortium 2003 ???).

Pode ser que o *Embaixador Utilizador* mostra-se cada terça-feira tarde para uma reunião de 90 minutos, e pode ser telefonou meia dúzia de vezes por semana. Isso geralmente é o suficiente para colocar o projeto para a zona de segurança. Se o utilizador não pode mesmo comprometer-se a uma hora por semana, então o projeto não está na zona de segurança, mas na verdade está em algum perigo em relação à adequação dos requisitos e adequação do feedback do utilizador.

O resto da equipas

Crystal Clear apenas nomes acima de três papéis, especificamente. Haverá outras pessoas no projeto. Como eles dividem o resto das funções do projeto é com eles. Em particular, alguém, ou várias pessoas, terá de capturar os requisitos em caso de uso ou forma de história de utilizador. Além disso, alguém vai ter que agir como coordenador. Talvez o Designer chumbo pode off-carregar este para alguém que tenha habilidade boa comunicação.

Fretamento: Executar os exploratórias 360 °

Em um ponto no início do projeto, a equipas - incluindo o patrocinador executivo - faz um uma vez em torno de verificação das questões-chave. Isto é o *Exploratórios 360 °* descrita anteriormente, ou um equivalente, tal como fase de "início" do RUP (Kruchten 2,002 ???). Neste cheque, eles se certificar de que o projeto não vai fundador em

- **o valor do negócio,**
- **os requisitos,**
- **o modelo de domínio,**
- **a tecnologia a ser utilizada,**
- **o plano do projeto,**
- **a composição da equipas, ou**
- **a metodologia ou convenções a serem usadas.**

Cada uma delas é analisada de uma forma de granulação grossa, para detectar se a equipas pretendido e metodologia projetada, usando a tecnologia destina-se em torno do modelo de domínio projetada, pode entregar o valor de negócio pretendido com o conjunto projetado de requisitos de acordo com o projeto de plano de projeto . Os exploratórios 360 ° resulta em um conjunto de ajustes para a configuração do projeto, ou no caso mais drástico, uma decisão do patrocinador executivo para cancelar o projeto (melhor agora do que mais tarde!).

Fretamento: Forma da metodologia

Moldar a metodologia muitas vezes pode ser feito em dois dias, e não deve demorar mais de uma semana para uma equipas Crystal Clear. Pode ser feito usando o *metodologia Shaping* técnica, casos de desenvolvimento do RUP (Kruchten 2003 ???) ou similares.

Lembre-se que a metodologia é nada mais do que as convenções a equipas se compromete a adotar. Uma vez que o conjunto de convenções será revisitada duas vezes por iteração ou ciclo de entrega, seria, em princípio, ser possível começar a partir de qualquer metodologia e simplesmente ajustá-lo ao longo do tempo para se adequar à equipas. Provavelmente não há tempo suficiente em um cristal típico

Limpar projeto para seguir essa estratégia. Será mais rápido para começar com melhor lista pensativo da equipas, de modo que as afinações subseqüentes não terá que ser tão drástica.

Como explicado em detalhe no *Agile Software Development,* há um custo bastante elevado no uso de uma metodologia excessivamente pesado: muito possivelmente que a equipas vai deixar de produzir software para o primeiro lançamento em tudo. Por esta razão, começar com um conjunto inicial mais leve de convenções do que se poderia suspeitar correta. Com menos regras, a equipas geralmente é mais provável para entregar software funcionando, e de reconsiderar as convenções, à luz da experiência, eles podem adicionar as convenções desaparecidas na oficina reflexão seguinte.

Fretamento: construir o plano de projeto inicial

Existem várias maneiras dentro dos limites de tolerância de Crystal Clear de construção de um plano de projeto base.

- **Meu próprio favorito é usar o *Planificação Blitz* técnica descrita anteriormente.**
- **O método mais rigoroso e cuidadoso que sei é do DSDM (DSDM Consortium 2003). Trata-se de um conjunto de verificações dos objetivos do projeto com o patrocinador executivo, seguido pela construção de um plano de lançamento e uma série de prazos de caixas, cada um a três meses de duração.**
- **método de planificação de Scrum corrige os time-boxes em um mês longa (Schwaber, 2001).**
- **No jogo de planificação do XP (Beck 1999, 2002), a equipas, incluindo o seu "cliente", escreve um conjunto de cartões de história para os requisitos funcionais. e classifica essas cartas em tempo-caixas de três semanas 39 (iterações). A equipas discute com o *Patrocinador executivo* a frequência adequada de liberação para os utilizadores reais, em seguida, agrupa os iterações em períodos de lançamento, muitas vezes cerca de três meses de duração. Cada uma dessas técnicas é dentro das tolerâncias metodologia de Crystal Clear. Cada um deles oferece um plano de projeto inicial, e é rápido o suficiente para que ele possa ser usado em uma base regular para monitorar mudanças na situação do projeto e faça um plano de projeto atualizado. Escolha um que você se encaixa.**

Reflexão sobre Fretamento

Para o projeto média, a equipas está sendo montada durante o período de fretamento. As pessoas que fazem o fretamento tem que lidar com uma série de variáveis. As pessoas podem ou não conhecer uns aos outros, ou podem aparecer em algum ponto no meio do projeto, necessitando de tempo para aprender a se comunicar uns com os outros.

Se o projeto emerge *Exploratórios 360* ° com um resultado positivo, a equipas tem uma noção de aonde o projeto é dirigido, o problema que está enfrentando, a tecnologia a ser utilizada, o plano do projeto, e seu valor de negócio. Isso cria um alinhamento antecipado de metas e ações que fortalece a equipas.

---

39 Na prática, a maioria das equipas XP parecem variar em qualquer lugar de uma a três semanas.

A formação metodologia e plano de projeto inicial deve levar um par de dias. Se os resultados têm de ser comunicadas a um público mais amplo, pode demorar mais alguns dias para escrever.

Todo o fretamento deve demorar alguns dias a uma semana para ser concluído, a menos que o projeto envolve uma nova tecnologia que é difícil de avaliar.

Quão importante é a ordem das etapas I esquema acima? Eu escrevi-los na ordem em que faz sentido para mim, mas eu não posso ver que a ordem é crítica. Escolha a ordem que faça sentido para você.

Os resultados da actividade fretamento são:

- **uma *ir* ou *não vá* votar sobre o projeto, e**
- **um grupo de pessoas com um projeto de carta e uma idéia do que eles estão fazendo.**

O ciclo de entrega

O ciclo de entrega tem três ou quatro partes:

- **A recalibração do plano de liberação**
- **Uma série de um ou mais iterações, cada um, resultando em integrada, testado código**
- **Entrega para os utilizadores reais**
- **Um ritual de conclusão, incluindo a reflexão sobre tanto o produto que está sendo criado e as convenções que está sendo usado.**

Recalibrar o plano de lançamento

No primeiro ciclo de entrega, deve, é claro, não ser necessário recalibrar o plano acabou de criar. Após a primeira entrega, no entanto, os membros da equipas vão encontrar-se com informações novas e valiosas para alimentar o plano do projeto. Eles terão aprendido tanto o quão rápido eles realmente funcionam e como equivocada suas estimativas de tamanho iniciais eram. Além disso, eles e seus utilizadores terão aprendido mais sobre o que é realmente necessário no sistema.

Eles têm uma escolha: eles podem ficar com o conjunto original de requisitos e apenas atualizar o plano, ou eles podem rever ambos os requisitos e o plano. O Crystal Clear não impor qualquer estratégia - que a decisão cabe ao patrocinador do projeto. Eles teriam, no entanto, ser negligente em seu dever profissional se não atualizar suas estimativas de trabalho e reexaminar as estratégias possíveis para o futuro.

Sou obrigado a repetir aqui a nota do *Planificação Blitz* técnica. Há uma interpretação do desenvolvimento ágil que o *Patrocinador executivo* está à mercê dos desenvolvedores na construção do plano do projeto. Esta interpretação é defeituoso. O que é verdade é que há uma alocação cuidadosa de responsabilidades na atividade de planificação. o

***Patrocinador executivo* controla as prioridades do projeto, a equipas de desenvolvimento controla a estimativa de quanto tempo cada tarefa vai demorar, eles *juntamente* compartilhar a responsabilidade para vir acima com um plano para entregar os melhores resultados ajustando o *Executivo Patrocinador do* prioridades com o tempo e as pessoas disponíveis. Se o *Patrocinador executivo* ainda sente que o projeto está demorando muito, ela tem exatamente três opções:**

- **substituir a equipas,**
- **ajuste de escopo ou tempo limites do projeto,**
- **venha com uma estratégia mais criativa para começar o trabalho feito com o tempo e pessoas disponíveis.**

Minha experiência é que uma combinação dos dois últimos trabalhos bem para entregar valor máximo negócios para recursos gastos.

iterações

ciclos de iteração está descrito na secção seguinte.

Entrega para os utilizadores reais

Tal como descrito em Entrega frequente, entrega em Crystal Clear significa "a um utilizador real." Em diferentes circunstâncias, isso pode significar a implantação completa com aulas de formação, ou muitos ser a de apenas uma pessoa, um amigável dispostos a dar o sistema de brotamento um passeio através, seja por curiosidade ou apenas como uma cortesia para a equipas.

O custo de implantação no primeiro caso, é muito caro, e assim por "entrega" provavelmente não pode ser feito a cada mês ou até dois, mas pode ter de ser feita trimestralmente. O segundo caso é barato. Minha experiência é que você pode geralmente encontrar um amigável, se você tentar; o valor obtido a partir desta pessoa é muito alta.

Muitos dos meus colegas, usado para implantação através da web ou para utilizadores amigáveis, internos não precisam de materiais de formação, me atacam regularmente por sugerir que uma equipas pode ir por três meses antes da implantação. Há três razões que eu aceitar a entrega em Crystal Clear para ser tão longo como, mas não mais de três meses:

- **As pessoas parecem ser capazes de manter seu foco de trabalho e memória detalhada por cerca de três meses, mas não visivelmente mais tempo. Em todas as minhas entrevistas do projeto, eu encontrei algumas equipas capazes de fazer ciclos de entrega trimestrais trabalhar de forma confiável, mas apenas um em quatro meses, e ninguém mais do que isso.**

- **O custo de entrega de um sistema pronto para uso em larga escala, com treinamento de utilizadores, é muito maior do que entregar o mesmo sistema para apenas um ou alguns poucos utilizadores amigáveis. O custo de tempo e dinheiro de testar o sistema, escrever os manuais, e criando o treinamento é suficiente que muitas vezes é impraticável fazê-lo mais do que trimestral.**

- **entrega trimestral de valor para o negócio é muitas vezes suficiente para que os executivos (a visão contrastante, é claro, é que a sua empresa pode-se tornar mais ágil se a entrega poderia ser feito mensal).**

Ritual completo: Refletir sobre a entrega

A pressão é normalmente bastante intensa em direção ao fim do prazo de entrega. Desenrolamento é parte de uma necessidade humana em geral para o ritmo. Fiquei surpreso ao ouvir três XP mestres falar sobre o burnout depois de um ano ou dois de prática bem sucedida XP. Eles disseram que seu trabalho estava operando a uma pressão constante, dia após dia, semana após semana, mês após mês. Não era que a pressão era alta, mas que era implacável. O mesmo foi relatado por um par de programadores usando Crystal Clear com iterações de duas semanas. Depois de um tempo, a monotonia da vida de trabalho estava ficando para eles. Em outras palavras, as pessoas precisam de uma chance para relaxar e "shake it out", para que eles possam crescer o novamente para o próximo período de intensidade.

Portanto, após a entrega do software, descontrair. Use o workshop reflexão como parte de "agitação-lo." Em vez de realizar um workshop de reflexão no final da última iteração antes da entrega, salve-o até logo após a entrega, em seguida, refletir.

Incluir uma celebração depois de um parto. Tenho ouvido de pessoas segurando uma festa, indo para uma caminhada nas montanhas, indo vela como um grupo, indo para casa mais cedo, ou alocação de alguns dias para as pessoas a trabalhar em uma nova tecnologia, algo diferente do sistema em desenvolvimento.

Um gerente providenciado para uma reunião fora do local de dois dias após cada uma das primeiras entregas (trimestrais). Eles passaram o primeiro dia na formação de equipas, socializar e refletir. Eles passaram o segundo dia planejando o próximo ciclo de entrega. Com exceção de "team building e socialização" (que leitores de alerta irá reconhecer atividades como essenciais no desenvolvimento ágil), ao mesmo tempo foi gasto em atividades do projeto como eles teria sido no escritório. A diferença era que tanto o lugar e o ritmo eram diferentes, de modo que a equipas voltou a seus escritórios em um estado refrescado. Depois de várias entregas, eles decidiram que não precisam de dois dias por mais tempo, e manteve a atividade combinada no local dentro de um único dia. No entanto, eles ainda tem o ritual de conclusão e mudança de ritmo que eles precisavam.

Depois de uma entrega, a equipa tem dois problemas adicionais para refletir sobre:

- **Como a implantação ir? Que ações diferentes devem ser tomadas no início do ciclo de entrega para reduzir a dor de implantação e treinamento?**
- **O que os utilizadores pensam do sistema? Quais são seus pontos fortes e pontos fracos? Mais importante ainda, pode a equipas de aprender alguma coisa sobre o que é *realmente necessário* pelos utilizadores, em comparação com o que foi originalmente solicitado.**

Refletindo sobre o processo de entrega é o mesmo que para qualquer outra oficina de reflexão: o grupo pede, O que nós gostamos e queremos manter o mesmo? E o que nós gostamos de fazer de diferente? A única diferença

é **que as mudanças afetam tanto hábitos de trabalho ao longo do ciclo ou atividades que devem ocorrer no início, talvez na primeira iteração de entrega.**

Para avaliar o sistema, a equipas pode se sentar e assistir os utilizadores, eles podem videotape alguns utilizadores usá-lo, eles podem deter grupos de foco no cliente, ou podem até mesmo contratar uma empresa externa para avaliar o sistema em uso. O ponto a ser feito é que é o *produtos,*
não o processo, para que reveja aqui.

O ciclo de iteração

comprimentos de iteração e formatos variam de acordo com diferentes equipas. Vou descrever dois ciclos de iteração possíveis, uma iteração de uma semana e uma iteração de dois meses. O seu é provável cair entre estes dois.

Uma iteração tem três partes:

- **planificação de iteração**
- **atividades diárias e integração de ciclo**
- **ritual de conclusão (oficina de reflexão e celebração).**

Dentro do período de iteração, a equipas irá encontrar-se adicionando à sua requisitos set, experimentar design de interface de utilizador, estendendo-se a infra-estrutura do sistema, adicionando funcionalidade, mostrando o sistema para o utilizador (s), acrescentando testes, e somando-se as capacidades de automação de seu ambiente de trabalho.

A uma semana iteração

No caso de uma semana de iteração, o planificação é provável de ocorrer na segunda-feira de manhã e a reflexão e celebração na noite de sexta-feira.

A sessão de planificação na segunda-feira de manhã é usado para definir as prioridades da equipas para a semana. Uma vez que uma semana não é muito tempo, a equipas deve dividir seu trabalho em bastante pequenos pedaços, a fim de garantir que eles possam desenvolver, testar e integrar as peças até sexta-feira à tarde. A boa notícia é que, uma vez por semana é um curto período de tempo, eles provavelmente não precisa gastar muito tempo estimar e analisar dependências. Espero que a sessão de planificação para a semana não levaria mais de cerca de uma hora na segunda-feira de manhã.

Cada manhã, ao meio-dia, ou no fim da tarde, a equipas é susceptível de ter seu diário *check-in 40* ou *Levante-se* encontro. Muitas equipas relatam que reuniões para cinco a dez minutos a cada dia aumenta a taxa de fluxo de informações dentro da equipas, e melhora a consciência de ambas as situações políticas e técnicas que podem se tornar relevante para eles os membros da equipas. A essência é a de manter a reunião curta (dez minutos) e deixar que cada membro da equipas sabe o que está acontecendo com os outros membros da equipas.

Durante a semana, além das reuniões anteriores, a equipas simplesmente opera em suas atividades de desenvolvimento normais. Seus episódios de desenvolvimento consistem em nada mais do que pegar uma atribuição de trabalho, desenvolvendo-lo e fazer check-in para o sistema de gestão de configuração, além de realizar um teste de construção de integração e do sistema, se essa é a convenção em uso na equipas. Eles vão discutir e, possivelmente, mostrar

---

40 Jim McCarthy escreveu um protocolo completo para as pessoas a registrar seus humores e intenções durante o projeto (McCarthy 2001 ???). o *check-in* protocolo é o utilizado para anunciar disponibilidade para trabalhar.

seu trabalho para o seu executivo utilizador patrocinador e embaixador de acordo com os ritmos que eles criaram.

O perigo para as equipas que usam iterações curtas (de um e dois semana) é que eles se esqueça de sempre trazer os utilizadores reais. Se esta for sua situação, considere a adição de um ciclo de super-que inclui visões de utilizadores, ou pelo menos criar uma explícita *Vendo utilizador* cronograma (ver *Produtos de Trabalho).*

ritual de conclusão para o one-week iteração

Para uma iteração apenas uma semana, cerimônia de encerramento da iteração é mais sobre o fornecimento de fechamento emocional do que qualquer outra coisa.

A síndrome de burn-out descrito na *Ciclo de entrega* é mais provável que aparecem quando o ciclo de iteração é muito curto. Como Ron Jeffries escreveu:

> **Outra coisa que eu gosto é o ritmo que varia de uma iteração. Ela começa com um plano, ele acumula-se ao longo de um par de semanas, em seguida, gira para baixo. (Se há um frenesi no final, não estamos fazendo certo ainda.) A iteração - modo livre, parece-me, seria apenas uma marcha inexorável marcha marchar. Fizemos isso em C3 por um tempo e ele nos levou louco, está me ouvindo, louco.**
>
> **Ron Jeffries 9/24/03 xp-egroup**

Com iterações mais longos, a integração final e a oficina de reflexão mais longo e longo sucedendo sessão de planificação fornecer uma forma de descompressão. Isso é menos o caso com iterações semanais, e assim uma série de jogos de computador, jogos de ping-pong, uma discussão clara sobre qualquer assunto profissional, um passeio de bicicleta, ou uma visita ao pub poderia estar em ordem.

Com uma iteração de uma semana de duração, não espere muito das oficinas de pós-iteração. Depois de algumas semanas, a equipas é improvável que seja capaz de pensar muito para mudar, e eles tendem a se tornar muito curto. Quando isso acontecer, considerar a realização da oficina de reflexão completo uma vez por mês ou dois, e bloquear uma hora inteira para ele, para que as pessoas possam refletir sobre o que aconteceu durante um longo período de tempo e Aviso repetir padrões que precisam de alteração. (Nota: isso cria o "super-ciclo" mencionado no início deste capítulo.)

Algumas equipas capturar sugestões durante as reuniões diárias de stand-up. Isto é excelente, mas não deve substituir a oficina de reflexão de fim de ciclo. Durante o stand-up diariamente uma sugestão para melhoria permite que a equipas de responder imediatamente. O workshop de fim de ciclo dá às pessoas um momento especial para refletir sobre o que eles acham positivo e negativo sobre os seus hábitos de trabalho. É por esta razão que um workshop mensal ou pós-parto reflexão pode ser de uma hora para metade de um dia longo.

A iteração de dois meses

O planificação para uma iteração dois meses pode tomar metade de um dia ou um dia inteiro, dependendo da técnica utilizada e da prática da equipas teve a fazê-lo. É comum para planejar a próxima iteração imediatamente após o workshop reflexão pós-iteração. Para aqueles sem uma técnica de planificação eficaz, eu recomendo começar com tanto o *Planificação Blitz* técnica neste livro ou XPS jogo de planificação (Beck 2002 Cohn 2004).

Durante a iteração, as mesmas atividades diárias ocorrem como para a uma semana de iteração: check-in diariamente ou stand-up, episódios de desenvolvimento, integração-build-e-teste, visitas com o utilizador eo patrocinador, e assim por diante.

Quando a iteração é de dois meses de duração, é útil realizar um workshop reflexão meados de iteração. Esta é mais curto e mais simples do que o workshop de fim de iteração. O objetivo principal desta sessão reflexão é descobrir se alguém detectou qualquer coisa que vai inviabilizar completamente o sucesso desta iteração. Se não, então talvez as pessoas têm encontrado algo que eles querem melhorar; ou então a reunião é apenas muito curto. Se eles descobriram um grande perigo, então, obviamente, eles devem descobrir o que fazer sobre isso. O objetivo dessas oficinas é pegar erros enquanto ainda há tempo para corrigi-los.

ritual de conclusão para a iteração de dois meses

A equipas pode usar uma hora para metade de um dia para o workshop reflexão se eles segurá-la a cada quatro a seis semanas. Eles devem ter o tempo para rever todos os aspectos do seu trabalho, de sua relação com seu patrocinador e os utilizadores, para os seus padrões de comunicação, hostilidade e amicability, a maneira como eles se reúnem os requisitos, as suas convenções de codificação, o treinamento que recebem ou não recebem , novas técnicas que pode querer experimentar, e assim por diante.

Após a primeira iteração ou ciclo de entrega, muitas vezes equipas apertar os padrões, obter mais formação, agilizar seu fluxo de trabalho, aumentar a testes, encontrar um "user friendly" e estabelecer convenções de gestão de configuração No final de ciclos subsequentes, suas mudanças tendem a ser muito menor, a menos que eles estão experimentando um radicalmente novo processo.

A maioria das pessoas pego em seu trabalho e não têm tempo ou inclinação para refletir sobre seus hábitos de trabalho. O ponto deste workshop periódica reflexão é pegar erros na hora de repará-los dentro do projeto e para dar às pessoas a chance de perceber padrões ineficazes.

No Japão, eles chamam estas sessões *Kaizen* oficinas 41. Veja um exemplo de saída a partir de um (não-software) oficina Kaizen no capítulo Produtos de Trabalho.

---

41 Estritamente falando, *Kaizen* é mais frequente e tipicamente tem uma forma e de saída específico. No entanto, mesmo no Japão eu não encontrei muitos grupos fazendo *Kaizen* em seu processo de software.

O Ciclo de Integração

Um ciclo de integração pode ser executado a partir de meia hora a vários dias, dependendo das práticas da equipas. Algumas equipas têm uma máquina stand-alone runn um script build-e-teste continuamente. Outras integram a cada poucos episódios de design, permanecendo em estreito contato com as atividades uns dos outros. Outros ainda integrar uma vez por dia ou três vezes por semana. Embora mais curto é geralmente melhor, Crystal Clear não legisla a duração do tempo de usar.

Para mais informações sobre a integração, reveja a propriedade, Ambiente técnica com testes automatizados, Gestão de Configuração e Integração frequente.

Seja usando minha oficina de reflexão ou o padrão *Kaizen* forma, a ideia é examinar e melhorar as convenções de trabalho com frequência.

A Semana e Dia

Dos vários ciclos, apenas os ciclos diários e semanais são ritmos de calendário. Muitas atividades de grupo ocorrem em uma base semanal. Estas podem incluir coisas como uma segunda-feira-manhã todas as mãos ou reunião de departamento, reunião relatório ou chefes de equipa, um regular discussão técnica trazer-seu-próprio-almoço ('saco marrom') seminário, ou uma tarde de sexta vinho-e partido - queijos ou busto cerveja (dependendo da sua cultura!).

Um dia de trabalho também tem o seu próprio ritmo. É provável que comece com uma reunião diária stand-up, em seguida, consistem em um ou mais episódios de design, com uma pausa para o almoço jogado em algum ponto.

Algumas equipas integrar o seu código várias vezes ao dia, caso em que o ciclo de integração não interagir muito com os ciclos diários e semanais. Outras equipas fazem uma compilação noturno, duas vezes por semana ou semanalmente.

O Episode Desenvolvimento

Ward Cunningham criou o termo *episódio* para descrever a unidade básica de trabalho programador no desenvolvimento ágil 42. Durante um episódio, uma pessoa pega alguns atribuição design pequeno e programas informáticos a conclusão (de preferência com testes de unidade), e verifica-lo para o sistema de gestão de configuração. Isso pode demorar entre quinze minutos e vários dias, dependendo do programador e as convenções do projeto. Mantendo o episódio menos de um dia de duração geralmente funciona melhor.

(Nota importante: Para mim, como para as muitas pessoas no mundo que trabalham no estilo de cristal, *desenho* e *programação* são tais atividades estreitamente ligados que eles não valem a pena erguer distante. Assim, não é apenas o papel de "designer programador" em Crystal Clear e Crystal Orange, não o designer dois papéis e programador.

Consequentemente, eu escrevo "para o programa", que significa "para projetar e programa" ou "para projetar", que significa "para projetar e programa". Eu escrevo "programador", que significa "programador designer-". Assim, no último parágrafo, a frase "pega uma atribuição design pequeno e programas informáticos a conclusão," significa pegar uma pequena missão que exige a concepção e programação, em seguida, projetar, programação, depuração e testá-lo, até a conclusão.

Eu preciso esclarecer isso porque muitos leitores pensam naturalmente de concepção e programação de ações de duas categorias de trabalho separados realizados por pessoas distintas. Isso seria uma má interpretação séria da Crystal Clear.)

42 http:? //c2.com/cgi/wiki DevelopmentEpisode ???

Reflexão sobre o Processo

Devido à história da nossa literatura, a maioria dos praticantes estão acostumados a ver uma versão linear do seu processo. A mudança mental para acomodar um processo cíclico é difícil. Não é que o *prática* de um processo cíclico é difícil - você provavelmente já funcionam em ciclos em seu projeto atual. No entanto, acho que mesmo os gerentes de projeto mais experientes e metodólogos não pode *explicar* processos cíclicos, e em especial as suas interacções de contorno (tais como a ausência de reordenamento no início do primeiro ciclo de fornecimento e a ausência de reflexão a oficina no final da última iteração de um ciclo de entrega). Para ser honesto, eu tenho tentado explicá-los por mais de dez anos e só agora sinto que tenho uma alça adequada sobre o assunto.

A dificuldade é que algumas atividades ocorrem em ritmos diários e semanais que se repetem sem criar qualquer sentido particular do "movimento para a frente." Estes *operações* tipos de actividades incluem reuniões de status diários e semanais, o check-in de código, executando testes de unidade, indo para o almoço, a convocação ao redor da máquina de café ou refrigerante. atividades de operações contrastam com atividades que mostram "progresso para a frente", tais como revisões de projeto, ficando requisitos sign-off, movendo-se em fases de teste ou versão alpha. As pessoas tendem a catalogar as atividades de progresso no processo e negligenciar as atividades de operações.

A visão de ciclo aninhado permite a discussão de progresso e atividades de operações.
Ambos são importantes para a vida do membro da equipas, ambos são parte do "processo".

Dois ciclos precisa de um pouco mais esclarecimentos: a iteração e a entrega. Na literatura recente do nosso campo, "iteração" é usado sem muita distinção. Ele poderia se referir a iterações estritamente internos, como eu descrevê-lo neste livro, ou para iteration- com entrega. Muitas pessoas esquecem sobre cumprir a parte de entrega do seu processo iterativo. Veja a discussão sob o título " *Nós colocated e correu iterações de duas semanas - por que nós falhamos"* no capítulo Misunderstood (erros comuns).

Em Crystal Clear, que estão autorizados a ter apenas uma iteração por ciclo de entrega, mas se você fizer isso, você *devo* ter algumas visões intermediários por utilizadores reais. Se você tiver várias iterações por entrega, alguns daqueles *devo* incluem sessões por utilizadores reais. Se você não fizer isso, você está construindo uma equipas de produção de software eficaz que torna o software errado de forma muito eficiente.

Capítulo 5 Examinado (Os Produtos de Trabalho)

Este capítulo descreve as funções da equipas e os produtos de trabalho, mostrando exemplos de cada produto de trabalho. Esses produtos de trabalho específicos são nem completamente necessário nem completamente opcional. Eles são os únicos que eu posso atestar tanto para um de cada vez e tomadas em conjunto. substituição equivalente é permitido, assim como uma quantidade razoável de costura e variação.

Embora este é o lugar aonde a maioria argumento é provável de ocorrer, é aonde o argumento é provavelmente menos susceptíveis de afectar o resultado do projeto.

Assim como descrever uma metodologia através do seu processo apresenta problemas de interpretação, o mesmo acontece descrevendo-o com os seus produtos de trabalho. Em uma pequena metodologia tais como Clear, o número e a formalidade de produtos de trabalho intermediários é reduzido de forma bastante significativa. A equipas vive da sua comunicação pessoal, notas sobre os quadros ou cartazes ao redor da sala, e demos ou entregas para a base de utilizadores.

No entanto, uma descrição dos produtos de trabalho é necessário. As pessoas apenas começando com Crystal Clear precisa ver o que conta como um conjunto de "aceitável" de produtos de trabalho. Executivos e patrocinadores precisa ver o que eles têm direito a pedir. Os professores precisam de um conjunto de produtos de trabalho que os alunos praticar. As equipas que estão fazendo muito pouco em termos de planificação e documentação precisa ver o que vale a pena preparar com até mesmo uma metodologia ágil luz. As pessoas que trabalham para entender uma metodologia vai querer examinar os produtos de trabalho como parte do pacote global metodologia.

Nesta seção, descrevo as funções e os produtos de trabalho. Esses produtos de trabalho deve ser visto como o padrão definido para um típico projeto de Crystal Clear, porque eles têm demonstrado o seu valor para muitos projetos. É, naturalmente, até a equipas do projeto para adicionar, subtrair ou modificar a lista com base em sua situação.

Cada item tem um propósito de comunicação no jogo económico-cooperativo. Às vezes, o artefato economicamente adequado é *baixa precisão (* não muito detalhado) e *em larga escala (* resumindo um grande tópico), sendo exemplos declaração de missão do projeto e o plano de lançamento. Em outras ocasiões, a *De médio precisão* artefato é necessário, como a lista ator-objetivo, que descreve os requisitos funcionais em um relance. As vezes, *alta precisão (*) descrições detalhadas são necessários, o código final, manual, e casos de teste sendo exemplos.

Há quase dois produtos dúzia de trabalho, dependendo de como você contá-los. Pessoas provenientes de um fundo tradicional projeto pode estar chocado com quão poucos existem; tingido-in-the-lã desenvolvedores ágeis estão chocados com quantos são (eles ficam chocados porque a menos que eles estão fazendo XP, eles simplesmente não têm contado quantos produtos de trabalho similares são produzidos em seus projetos atuais). Os produtos de trabalho aqui são luz e distribuído entre os papéis, para que a equipas não deve encontrá-los onerosa na prática. Como um exercício, sugiro contagem do número total de produtos de trabalho gerados no seu projeto atual. A minha experiência é que o número raramente é inferior a 60. Compare-os com os listados aqui.

Surge a pergunta sobre produtos de trabalho mais do que por qualquer outro aspecto de uma metodologia: "Nós temos que fazer *tudo* destes?" A resposta é difícil de dar, como ela se encontra em algum lugar entre a resposta que eu era capaz de dar para Propriedades ('Por Crystal Clear, absolutamente os três primeiros, e tanto os próximos quatro possível.') eo que eu era capaz de dar de Técnicas ("Completamente a critério da equipas; aqui estão um motor de arranque interessante e útil definir a considerar.").

Se você pular muitos desses produtos de trabalho, você perde o alinhamento e visibilidade. Sua orientação e apoio pode sofrer em conformidade. Por outro lado, seria apenas

tolo para me insistir que você faz todos eles como dado - os produtos de trabalho específicos necessários por equipas varia de acordo com técnicas de mudança, tecnologias, hábitos de comunicação e até mesmo moda.

Pediram-me para nomear "os principais" produtos de trabalho, como eu era capaz de nomear propriedades "o núcleo", mas isso não é possível. Se eu estava indo para escolher um produto de trabalho a cair, seria o Mapa de Projetos (por mais útil que possa ser). Se eu estava indo para escolher um outro, seria o cronograma de visualização (que poderia ser feito verbalmente e em tempo real). Se eu tivesse que cair outro casal, eu poderia sugerir fusão do plano de iteração de granulação grossa e de granulação fina e status e soltando o plano de iteração de granulação fina. Poderia acontecer que outra pessoa iria deixá-los em outra ordem, ou eu poderia deixá-los em uma ordem diferente em um projeto diferente.

Desalinhamento dentro da equipas e falta de comunicação com o mundo exterior crescer a cada omissão. Os riscos se acumulam, lentamente no início, até que o projeto não está na zona de segurança, e nunca é claro quando ele deixou de ser seguro para ser inseguro.

É **muito fácil sobrecarregar um projeto com o trabalho que é útil individualmente, mas que, quando tomados em conjunto, retarda o grupo a ponto de perder mais segurança do que ele adiciona. Este conjunto de produtos de trabalho é composto por itens que posso defender individualmente e todos juntos. Mesmo que sua equipas faz cada um deles, ele não deve ser overburdensome. Você ainda pode ser capaz de encontrar uma maneira de substituir ou omitir alguns deles. Discuti-lo em sua oficina metodologia de modelagem, e discuti-lo novamente em suas oficinas mensais ou trimestrais reflexão. Isso é o que aqueles são.**

Nas páginas que se seguem, eu descrevo cada produto de trabalho, indicando qual o papel é responsável por isso. I incluem vários exemplos de cada produto de trabalho para mostrar variações de estilo e formato diferentes equipas adotaram. Sempre que possível, eu mostro representações formais e informais, gráfica e textual. Vendo esse intervalo, você pode escolher uma forma que funciona para você e os grupos externos que você precisa para se manter informado.

As funções e os seus Produtos de Trabalho

o Patrocinador é responsável por produzir apenas um item:

a *Missão com tradeoff Prioridades.*

o Equipas como um grupo é responsável pela produção de duas coisas:

a *Estrutura equipas e Convenções,* e a *Resultados*
reflexão oficina.

o coordenador, com a ajuda da equipas, é responsável pela
produção a *Projeto Mapa,*
a *Plano de lançamento,*
a *Status do projeto,*
a *Lista de Riscos,*
a *Iteração Plano & Status,*
a *Visualizando agendamento.*

o Business Expert & Embaixador Utilizador juntos são responsáveis pela
produção a *Lista Ator-Meta,*
a *Casos de Uso e Requisitos de arquivo,* e a *Modelo função*
do utilizador.

o Designer-chefe é responsável pela produção do
a *Descrição da Arquitetura.*

o Designer-programadores (incluindo o Designer de chumbo) são responsáveis pela
a *Rascunhos de tela,*
a *Modelo de Domínio comum*
a *Design Sketches & Notes,*
a *Código fonte,*
a *Código migração,*
a ***testes,***
a ***Embalado Sistema.***

o Tester (quem está ocupando esse papel no momento) é responsável pela produção

a *Relatório de erros* naquela hora. o Escritor é
responsável pela produção
a *Ajuda ao utilizador de texto.*

funções: **Patrocinador, Embaixador do utilizador, Designer Chefe, DesignerProgrammer, Business Expert, Coordenador, Tester, Escritor**

Há oito papéis nomeados para Crystal Clear, quatro dos quais provavelmente têm que ser pessoas distintas. Os outros quatro podem ser funções adicionais atribuídos a pessoas no projeto. Os primeiros quatro pessoas são:

- **Patrocinador executivo**
- **Embaixador Utilizador**
- **Designer-chefe**
- **Designer-Programmer**

Patrocinador executivo. Esta é a pessoa que é ou atribuição ou defender a destinação do dinheiro para o projeto. o *Patrocinador executivo* é suposto para manter a visão de longo prazo em mente, equilibrar as prioridades a curto prazo com os de versões subseqüentes e equipas subseqüentes em evolução ou manutenção do sistema. Esta
é **a pessoa que irá criar visibilidade externa para o projeto, e fornecer a equipas com decisões cruciais de nível empresarial. Se há uma questão de equilibrar o dinheiro a ser gasto para o valor a ser entregue, cabe à *Patrocinador executivo* para tomar a decisão de quando e se quer continuar ou parar, e se continuar, como aparar as funções do sistema restantes para recuperar o valor do negócio. Parte da metodologia envolve dar a**

Patrocinador executivo **boas informações para tomar essas decisões.**

Às vezes é a comunidade de utilizador que está realmente pagando, ou *patrocinadora,* o projeto, e eles, como um departamento, trabalhar através de um *executivo,* que dirige a equipas de desenvolvimento. É fácil nesta situação para o *Patrocinador*

executivo **papel para se desajeitadamente dividido, aonde o executivo não tem as mesmas a longo prazo interesses, prioridades ou visão como a comunidade de utilizadores. Não há nada que uma metodologia como Crystal Clear pode fazer para remover o conflito potencial aqui. Você, como um grupo combinado de pessoas, têm de reconhecer os perigos desta situação e trabalhar duro extra na comunicação para manter prioridades e metas alinhados, visibilidade e amicability alta.**

Embaixador utilizador. Esta é a pessoa que é suposto estar familiarizado com os procedimentos operacionais e o sistema em uso (se já existe um), sabendo que são com frequência - e raramente - modos de operação, o que atalhos são necessários usado, e quais informações tem de ser visível juntos na tela ao mesmo tempo. Esta é uma base de conhecimento diferente do que o *Business Expert* deverá ter. A pessoa que segura o papel de *Business Expert* Espera-se que conhece as regras empresariais necessárias dentro do sistema, o que o negócio políticas são
estáveis contra que são susceptíveis de mudar. *Businesso Expert* Não é esperado que têm íntima familiaridade com as atividades minuto a minuto da população usuária, enquanto o *Embaixador Utilizador* é.

Designer-chefe. Esta é a pessoa técnica chumbo, a pessoa deve ter experiência com desenvolvimento de software, capaz de fazer o design de sistema principal, dizer quando

a equipas do projeto é na pista ou fora da pista, e se fora da pista, como voltar para pista. Em termos de níveis de competência, a *Designer-chefe* é esperado para ser um designer de nível 3 43.

Há argumentos sobre se o desenvolvimento ágil requer programadores de alto nível para ter sucesso. Eu nunca tive o luxo de usar apenas os programadores de topo, nem foram as equipas que entrevistei compunham deles. Eles foram feitas pela habitual mistura de pessoas encontra-se em uma empresa. É importante notar, no entanto, pelo menos *1* pessoa da equipas foi competente e experiente, isto é, o nível 3.

A proporção de nível 3 pessoas para nivelar-1 pessoas é um fator que considero significativo o suficiente para que eu oferecê-lo aos pesquisadores para examinar como um fator de correlação principal para o sucesso do projeto. A minha experiência (e eu suspeito que a pesquisa vai mostrar) que isso se aplica a ambos os pequenos e grandes projetos, ambos os projetos de baixa e alta cerimônia. cerimônia de alta precisam de muito mais clichê trabalho feito, e assim pode viver com uma maior mistura de pessoas menos talentosos para os talentosos. Barry Boehm e Rich Turner captar essa diferença em seu diagrama de estrelas do mar (*Equilibrar agilidade e disciplina,* p. xxx ??), em que um dos eixos é rácio de nível-3 ao nível 1 pessoas.

Uma vez que existem apenas três a oito pessoas em um projeto de Crystal Clear, pelo menos um deles tem que ser competente e experiente, ou seja, o nível 3. Normalmente, há um par de estagiários ou pessoas júnior no projeto. Para efeitos de segurança do projeto, então, eu designar o *Designer-chefe* papel separadamente dos outros papéis.

o *Designer-chefe* é susceptível de ter grande influência na oficina metodologia de formação. Muitas vezes, o *Designer-chefe*

é o único designer experiente na equipas. Se toda a equipas é experiente, então é claro que a oficina shaping metodologia pode trabalhar de forma peer-to-peer.

Eu uso a palavra "designer" para este papel para encurtar o nome da função do excessivamente longo " *Chumbo Designer-Programmer."* o *Designer-chefe* Espera-se que tanto o design eo programa, assim como são os outros *Designer-programadores.*

Normalmente, mas nem sempre, o *Designer-chefe* é a pessoa mais experiente da equipas e também lida com a atribuição de programação mais difícil. Embora isso é comum, ele realmente representa um risco para o projeto. o *Designer-chefe* tende a ficar sobrecarregado, tendo o papel como *coordenador* bem como arquiteto, mentor e programador mais

experiente. Qualquer coisa que pode ser feito para descarregar o *Designer-chefe* é susceptível de melhorar o perfil de risco do projeto. A escolha óbvia é ter alguém parte do jogo o mais ou todos os *coordenador* Função.

Designer-Programmer. I mesclar as palavras "design" e "programador" na

Designer-Programmer papel para destacar que cada pessoa ambos os projetos e programas. Projetando sem programação está cheia de falhas devido à falta de feedback em projetos de

43 Lembre-se do Capítulo 1, Nível 1 de habilidade é a fase "seguindo os procedimentos" de aprendizagem; Nível 2 é "romper com procedimentos específicos" e o nível 3 é "fluência", ou mistura e inventar na mosca.

todos os tamanhos. É muito particularmente não tem lugar em um projeto do tipo Crystal Clear. Programação por sua própria natureza envolve concepção. Nem "designer" nem "programador" está sozinho como um nome de função, portanto, *Designer-Programmer.*

Os outros papéis. o *coordenador* é provavelmente uma ocupação parcial para alguém da equipas; projetos de apenas quatro a oito pessoas raramente têm um gerente de projeto dedicado. Pode haver uma pessoa gerenciar vários projetos, e tocando o *coordenador* papel para cada um deles. Alternativamente, a *Patrocinador executivo* ou *Designer-chefe* pode obter esse papel, ou mesmo que as pessoas se revezam segurando esta posição.

A pessoa que ocupa *coordenador* deve, no mínimo, tomar notas durante as sessões de planificação e status do projeto, e pentear as informações para a publicação e apresentação. o *coordenador* é responsável por dar a visibilidade patrocinadores do projeto na estrutura e status do projeto. Com sorte, o *coordenador* Também é alguém com um bom toque humano, e pode facilitar as discussões e reduzir conflitos.

o *Business Expert* é o especialista de como o negócio funciona, quais as estratégias ou políticas são fixos, o que é susceptível de variar em breve, muitas vezes, ou raramente. Essa pessoa vai responder a todas as perguntas variadas

os desenvolvedores irão ter sobre o coração do sistema. É informação diferente do que o *Embaixador Utilizador* normalmente fornece, embora em alguns casos, pode acontecer que o *Embaixador Utilizador* é também o *Business Expert.* Em várias situações, eu vi o *Patrocinador executivo,* a *Designer-chefe* ou o *Embaixador Utilizador* ser o *Business Expert.* Às vezes uma pessoa externa é trazido para que a perícia.

***Testador* e *Escritor* são susceptíveis de ser rotativa ou trabalhos temporários. I dar a estes nomes dos papéis separados, mas em muitos projetos Crystal Clear há apenas os quatro a oito pessoas, todos os programadores, sentado na sala. Neste caso, eles, obviamente, tem que se revezam ocupando esses papéis. Algumas equipas podem começar o uso de um *Escritor* por períodos de tempo, ou ter um dedicado *Testador* trabalhando e mesmo sentado com eles.**

Uma nota sobre as amostras do projeto

No seguinte, eu desenho a partir de um conjunto de projetos, alguns, mas não todos os quais foram projetos Crystal Clear.

o *PRTS* projeto foi o projeto de construção de um sistema para rastrear todos os pedidos de compra emitidos por funcionários do Banco Central da Noruega, se eles foram cumpridos a partir do armazém interna ou fornecedores externos. Este projeto, pela sua curta vida, consistia de quatro pessoas que trabalharam nele exatamente um dia a cada semana. Reuniu-se o Comunicação osmótica exigência, porque ninguém trabalhou com ele em tudo, durante quatro dias, e então nós trabalhamos juntos no quinto dia de cada semana.

Eu era o *coordenador* para este projeto. Procedemos de declaração de missão, através *Blitz Planificação,* e através do *Exploratórios 360 °*. Durante o *Exploratórios 360 °,* o projeto proposto não tanto o os testes de avaliação de negócios pico tecnologia e. Os programadores queria continuar nele, mas o *Patrocinador executivo* decidiu comprar um pacote para o sistema, a fim de liberar os programadores para um trabalho mais importante para o negócio. Não só isto foi a decisão correta negócios 44, mas os programadores encontrado o seu próximo projeto muito mais emocionante, uma vez que envolveu tanto o valor de negócio crítica e nova tecnologia.

Consequentemente, há produtos de trabalho para PRTS para cima através desse ponto. o *NICS-NBO* projeto. Devido a uma falha de banco no início de 1990, todos os bancos noruegueses tem que manter uma "conta corrente" com o Banco Central, e todas as transações bancárias bancárias-a-ponto tem que se canalizados através destas "contas correntes." A união bancária, representantes de bancos-chave, eo gerente da divisão de Banking do Banco Central compreendeu o comitê de direção para uma série de projetos. O projeto NBO aqui era o terceiro da série. Eu fui *coordenador* para este projeto.

No início, foi aberta com os mesmos três programadores que tinham feito os dois primeiros projetos. Parecia um candidato fácil para Crystal Clear, mesmo que a tecnologia era de mainframe com COBOL, Assembler, CICS e LU6.2. Com o tempo, nós perdemos o *Designer-chefe (* licença de paternidade), metade da segunda pessoa sênior (projetos Y2K) e ficaram apenas com o desenvolvedor júnior e uma nova contratação. A equipas depois cresceram em número e perdeu Comunicação osmótica. Neste ponto, o projeto estava fora de Crystal Clear gama. A amostra do produto trabalho que estou usando aqui é o plano projeto- de grão grosso e a combinação de uma única folha de status e de risco lista que usamos para relatar ao comitê de direção a cada mês.

o *BSA* projeto. Este foi um projeto realizado como parte de um curso último ano na Universidade Estadual de Weber. A pequena equipas reuniu-se uma vez por semana fora de classe e comunicada por telefone e e-mail de outra forma. Eles usaram tecnologia de banco de dados para o

44 Niel Nickolaison (2004) escreveu um procedimento de avaliação de negócios simples para decidir quais os projetos a desenvolver em casa e que terceirizar, consulte http: // ???

projeto, para o modelo de domínio consiste de um esquema de banco de dados. Eu gostaria de agradecer ao Dr. Christopher Jones e esta equipas, Eric Schultz, Keith Deppe, Tony Hess, e de Betânia Stimpson para este projeto.

Projeto *Winifred.* Projeto Winifred é o 50-pessoa, US $ 15 milhões, 18 meses preço fixo, projeto em tempo fixo I descrito em pormenor no *Sobrevivendo Projetos Orientados a Objetos,* e para o qual nós construímos o original Crystal Orange. Eu uso o mapa do projeto e plano de entrega de granulação grossa desse projeto, porque parece interessante para mim que o nosso projeto de 18 meses tinha um plano de granulação grossa composta por cerca de oito bolhas (apoiados por cerca de 240 casos de uso de dois parágrafos).

o *Calibração de câmera CamCal* protótipo. Este foi um ensaio de Crystal Clear na Thales, uma organização certificada ISO 9001. Este projeto envolveu quatro pessoas por quatro meses e se encaixam as diretrizes cristalina. Eles foram tipo suficiente para deixar me incluir seu relato de experiência e recomendações do auditor neste livro (veja o capítulo,
Testado).

Projetos adicionais estão representados nas amostras de produtos de trabalho. Estes, no entanto, são descrições de amostra dos tipos de projetos utilizados.

***Patrocinador:* Missão com tradeoff Prioridades**

A declaração de missão é uma descrição breve, normalmente um parágrafo a uma página, do que está a ser construído, o seu propósito em um contexto maior, e as prioridades do projeto em sequência, desde o mais crítico para aqueles que podem ser sacrificado.

É **produzido pela *Patrocinador,* Antes que o projeto for iniciado ou durante *fretamento projeto.* Ele é analisada pela *Business Expert* e *Designer-chefe,* e referenciado por todos na equipas. Qualquer grande mudança na declaração de missão desencadeia uma reunião da equipas, para que todos se a equipas entende a nova missão.**

Uma boa declaração de missão mantém a sua clareza e relevância ao longo do tempo, mantendo os esforços de trabalho centraram-se sobre as características mais importantes do sistema.

Porque as pessoas geralmente podem proteger apenas um ano e possivelmente duas prioridades do projeto (ver *Peopleware,* pp. ??), não tem mais de dois itens prioridade. As prioridades podem ser escolhidos a partir de: bater uma data de entrega particular, custo, facilidade de uso, facilidade de aprendizagem, a velocidade em uso, exatidão, desempenho, facilidade de manutenção, flexibilidade de design, proteção de responsabilidade legal (você pode vir até com outras opções para seu projeto). A declaração

de missão torna claro para a equipas que pode ser sacrificada se necessário, para preservar as principais prioridades. Faço notar aqui que é improvável ter sucesso em ambos entrega rápida e exatidão como as duas principais prioridades. Esses são dois que precisam ser negociadas uns contra os outros.

Eu forneço três exemplos aqui, do projeto de Calibração da câmera, os escoteiros programa de rastreamento eo projeto PRTS, respectivamente. Veja também *Agile Project Management* (Highsmith 2004) para exemplos.

CamCal Camera Missão Calibração

A missão do projeto calibração da câmera é:

- **Desenvolver uma ferramenta de software de fácil manutenção, simples de usar, portátil e extensível que permitirá vídeo inteligente instaladores e reparadores de sistema de vigilância para rapidamente, com precisão e eficiência calibrar suas câmeras do sistema com respeito a uma pré-definido modelo de cena tridimensional.**

- **Estabelecer uma base para o desenvolvimento de um conjunto de ferramentas futuro que pode gerenciar todos os aspectos das atividades**

  de instalação do sistema de vigilância de vídeo e oficinas de reparação inteligentes. Tal conjunto de ferramentas irá incluir a ferramenta de calibração de câmera, e irá adicionar capacidades de criação de modelo de cena. Ele vai ser capaz de crescer com o crescimento da tecnologia do mercado de vigilância por vídeo.

- **Desenvolver e experimentação do uso de metodologias e processos novos de desenvolvimento de software, com vista a melhorar o processo de desenvolvimento de software usado para pequenas e projetos de software de médio porte.**

As prioridades de desenvolvimento são:	
Sacrificar os outros para isso:	**Concluir até final de Janeiro, garantir a precisão e qualidade**
Manter, se possível:	**Usabilidade, potencial para crescer em uma ferramenta mais extensa**
Sacrificar estes primeiros:	**velocidade de execução, portabilidade, interfaces adicionais, armazenamento formato interno.**

Figura 5-1. declaração de missão CamCal. (Graças a Stephen Sykes na Thales)

O "BSA Progress Tracker" é projetado para scoutmasters para acompanhar o andamento dos escuteiros através de seu avanço na Boy Scouts of America programa para Eagle Scout.

Prioridades de desenvolvimento

abrangente: Data de entrega, funcionalidade

Restrições de primeiro plano: Facilidade de uso, correcção formal

Cordeiro sacrificial: atuação

Figura 5.2. declaração de missão BSA.

A "Compra Request Tracking System" tem dois objetivos. O primeiro e mais essencial é fornecer um sistema básico para os compradores oficiais da empresa para acompanhar o que eles têm ordenados de fornecedores contra o que foi entregue. O segundo é o de simplificar a vida das pessoas que desejam pedir coisas, que deverão assinar pedidos de compra, e quem são para rastrear as compras contra orçamentos.

Prioridades do projeto:

	Sacrificar os outros para esta	Tente manter Sa	crifício estes para outras
Simples de usar	X		
Baixo custo para desenvolver	X		
liberdade defeito		X	
entregar em breve		X	
Facilidade de aprendizagem			X
atuação			X
flexibilidade de design			X

Figura 5-3. declaração de missão PRTS.

Equipas: **Estrutura e Convenções equipas**

estrutura da equipas é uma alocação de pessoas para papéis. convenções da equipas é um conjunto de regras, práticas e convenções a equipas se compromete a adotar.

convenções da equipas mudar constantemente. Este é o caso, porque a situação em que a equipas se encontra está mudando constantemente. A equipas deve não apenas estar consciente sobre essas mudanças, mas deve deliberadamente procurar o melhor conjunto de convenções para seu projeto.

A parte publicada do Crystal Clear contém um grande número de acordos e convenções para a equipas a adotar. Sendo escrito para muitos projetos, não pode ser completa. A equipas precisa resolver convenções adicionais, pelo menos o seguinte:

- **alocação de pessoas para papéis,**
- **convenções de programação como nomear, formatação, comentando,**
- **convenções de propriedade de código,**
- **projeto e código de revisão convenções, quer revisões de código ou programação em pares ou nenhum,**
- **duração da iteração e formas de relatório de status, a frequência de visualização do utilizador,**
- **convenções de gestão de configuração.**

Dependendo da experiência da equipas, as convenções podem ser construído através de uma oficina de metodologia de conformação aberta, ou elaborado inicialmente pela *Designer-chefe* e permitiu a evoluir por sugestão e consenso de toda a equipas em workshops de reflexão ("consenso" significa que cada pessoa está disposta a aceitar e defendê-la, mesmo que isso não é preferência pessoal dessa pessoa). Algumas convenções exigem consenso da

Patrocinador **também.**

As convenções devem ser conscientemente revisada na oficina reflexão pós-iteração, ou pelo menos a cada 3 meses. reler o *iteração Ciclo* a analisar este. A qualquer momento, é claro, alguém da equipas pode descobrir que algo realmente não está funcionando, e pode solicitar uma alteração à forma como a equipa está a trabalhar.

convenções da equipas são muitas vezes falado em vez de escrito. Alguns podem ter escrito sobre os radiadores de informação.

Codificação convenções são frequentemente exibidos por meio do código de amostra.

I categorizar uma metodologia como "bem sucedido" se

5. **o software fica a entregue e**
6. **a equipas está contente de trabalhar da mesma forma novamente.**

Isto pode parecer ser um limite muito baixo para o sucesso, mas os relatórios recentes do projeto mostrá-lo para ser surpreendentemente difícil.

Se a sua metodologia de falhar em qualquer contagem, alterá-lo usando a técnica de oficina de formação metodologia.

A estrutura da equipas no Crystal Clear deve ser bastante simples para estabelecer. Quatro papéis são cruciais: *Patrocinador Executivo, Designer Chefe, Designer-programadores,* e *Embaixador utilizador.* Os outros papéis, *Business Expert, Coordenador, Testers,* e *Escritoras* pode

ser combinados papéis atribuídos, quer para as pessoas adicionais ou compartilhados por pessoas que têm outros papéis. Muitas vezes, o *Patrocinador Executivo, Designer de chumbo* ou *Embaixador Utilizador* é o *Business Expert.*

Aqui está um exemplo de atribuição de pessoas para papéis.

Nome do time:	**BSA-Team**
Patrocinador:	**Eric Schultz**
Designer-chefe:	**Keith Deppe**
Do utilizador:	**Eric Schultz**
Designer-Programmer:	**Tony Hess Betânia Stimpson**
Business Expert:	**Keith Deppe**
coordenador:	**Betânia Stimpson**
Testador:	**Tony Hess**
Escritor:	**Betânia Stimpson**
Caminhos de comunicação:	**Reuniões semanais fora de classe. Telefone e e-mail comunicação.**

Figura 5-4. atribuições de função BSA

Um exemplo simples de uma convenção adotada por uma equipas do departamento de teste segue. Eles estavam tentando passar de manual para testes automatizados, mas correu para o problema que eles passaram a maior parte de seus fogos tempo de combate (usando testes manuais), e teve problemas para obter-se a sentar e escrever testes automatizados. Eles, portanto, estabeleceu para si a convenção:

" Nenhum teste manual antes de 3 pm"

Esta regra simples deu-lhes a Focus (tanto prioridade e tempo) que precisavam para superar os obstáculos iniciais e construir um conjunto inicial de testes de regressão automatizados.

Aqui está um trecho de algumas convenções da equipas que criamos para uma empresa de 50 pessoas (50 pessoas coloca-o na categoria Crystal Orange). Havia várias convenções dúzia, que nós dividido em cinco categorias: pulsação regular com a aprendizagem; Processo Básico; Máximo Progress & mínimas distrações; Maximamente livre de defeitos;

A Comunidade Alinhados na conversação. I listar todo o conjunto de convenções em *Desenvolvimento Ágil de Software.* Por razões de espaço, eu incluir aqui apenas a terceira seção como ilustrativa.

Progresso, Distractions Máximo Mínimo

O objetivo desta categoria é para garantir que as pessoas estão trabalhando no que é de maior valor para a empresa e ter tempo para se concentrar e fazer progressos nesse trabalho.

1. As principais iniciativas empresariais de topo são priorizados e visivelmente afixado para cada dois ciclo de produção semana.

2. Eles são atribuídos a pessoas individuais para que cada pessoa conhece o seu topo dois ou três itens prioritários pessoais para o ciclo.

3. O trabalho está dividido no que pode ser concluído e testado nos ciclos de duas semanas e é ainda dividido em coisas que podem ser realizadas em um a três dias úteis.

4. Cada pessoa que está trabalhando em mais do que uma iniciativa é garantido pelo menos dois dias consecutivos para trabalhar em qualquer uma iniciativa sem ser puxado para outra tarefa.

5. Os desenvolvedores postar nos quadros fora do seu escritório o estado actual do o trabalho que pretende concluir durante uma determinada semana.

6. Todas as manhãs, os desenvolvedores se reunir com o proprietário da empresa do trabalho atual iniciativa e realizar uma breve reunião para determinar o estado atual do trabalho e as prioridades de trabalho superior e para discutir quaisquer dúvidas.

7. O proprietário da empresa não é permitido para pedir o status novamente o resto do dia.

8. O período de 10:00-12:00 a cada dia é declarado "tempo de foco", em que há reuniões ter lugar, e todos na empresa é encorajada a desligar o telefone.

Figura 5-5. convenções do projeto. (Graças a eBucks.com)

***Equipas:* Resultados reflexão Oficina**

A reflexão Workshop de Resultados é um radiador de informações que mostram o que a equipas tenha concluído após a sua oficina de reflexão. Muitas vezes, é um flipchart contendo "As coisas que devemos manter", "coisas para tentar", e "problemas em curso" (veja *Workshop de reflexão,* no capítulo 3). O quadro está pendurado em um lugar altamente visível para que a equipas pode perceber o que convenções e práticas são o tema da iteração.

A saída é produzido por toda a equipas durante ou na conclusão da oficina reflexão. O feedback dos times indica que ele não é uma boa idéia para dar uma pessoa a tarefa de "limpar" o flipchart utilizado na oficina: Para a primeira parte, que a pessoa geralmente muda as palavras na carta, para que ele não o é palavras que receberam aprovação da equipas. Para a segunda parte, o "limpo" versão simplesmente parece diferente do que as pessoas recordam, e assim eles não se identificam com ele também. De modo geral, não importa muito se o gráfico é um pouco confuso desde que seja legível. Se você estiver indo para limpá-lo, fazê-lo como parte do workshop.

A resultados do workshop gráfico boa reflexão é grande, visivelmente colocado, e mostra a equipas que eles devem prestar especial atenção ao nas próximas semanas. Assim, é melhor usar a frase "Try This" (que afirma concretamente o que todos deveriam experimentar) em oposição a "Needs Improvement" (que lembra a todos do que
é ruim sem sugerir um remédio).

Aqui estão alguns exemplos.

Figura 5-6 (Graças a Tomax)

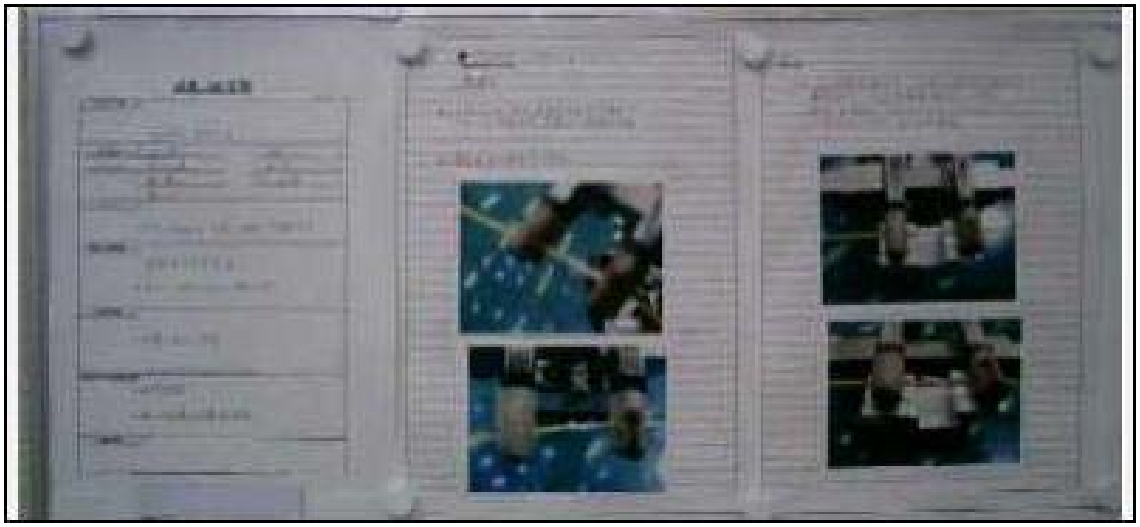


Figura 5-7. Kaizen gráfico de saída do Japão (Graças a Basaki Satoshi)

coordenador: **Projeto Mapa, plano de lançamento, status do projeto, Lista de Riscos, Plano & Status iteração, Visualizando Cronograma**

O patrocinador, a equipas de desenvolvimento e os utilizadores toda a necessidade de ser capaz de planejar suas atividades em torno de quando as funções do sistema são supostamente para tornar-se visível e disponível. A equipas precisa saber a ordem em que para criá-los e as datas-alvo, os utilizadores precisam saber quando eles devem alocar tempo para avaliá-los ou começar a usá-los.

projetos Crystal Clear operar com dois horizontes temporais distintos, de longo prazo e curto prazo. A longo prazo horizonte normalmente se refere a qualquer coisa ao longo de três meses, embora possa ser usado para qualquer coisa mais do que uma única iteração. Planificação e acompanhamento para o horizonte de longo prazo é *granulação grossa.* Isso permite que a equipas para fornecer os executivos com as estimativas em recursos naturais e expense- necessários para o planificação global da empresa, tendo em conta que a equipas não vai saber o que os custos reais será até que eles ficam mais adiante.

A curto prazo horizonte tipicamente refere-se a uma única iteração. *Refinadas* planificação e monitoramento é utilizado para o horizonte de curto prazo. Isso é para que todos possam ver o progresso da equipas e detectar problemas rapidamente. O plano de curto prazo e status são geralmente capturadas em radiadores de informação na sala de equipas.

Esses produtos de trabalho são produzidos em diferentes épocas e mantido em maneiras diferentes. Estes são o

- **mapa projeto**
- **lista de riscos**
- **visualização e liberar cronograma (o plano de grão grosso)**
- **status do projeto (grosseiro)**
- **plano de iteração (o plano de grão fino)**
- **status de iteração (refinado)**

Como mencionado anteriormente, pode haver uma pessoa dedicada atuando como *coordenador* sobre o projeto, mas mais frequentemente é o *Patrocinador* ou *Designer-chefe* que assume esse papel. Alguns grupos acham que é útil ter alguém para assumir o *coordenador do* papel, para reduzir a carga sobre o normalmente sobrecarregado *Designer-chefe.*

coordenador: **Mapa projeto**

O mapa do projeto é um diagrama de dependência identificar o principal trabalho a ser feito e quais depender dos outros (ver a *Planificação Blitz* técnica). O seu objectivo é mostrar a estrutura do problema e a sequência de ataque. Ele não contém datas. Isso é em parte porque muitas vezes é elaborado antes dos membros da equipas e suas capacidades são conhecidos, mas de modo mais geral, para permitir diferentes estratégias de pessoal e de tempo a ser considerado. Destina-se a ser visto de uma só vez, ou em uma mesa da sala de conferência ou em uma parede, usando cartões de índice como marcadores de tarefas. Isso limita o número de itens que são colocados no mapa.

O mapa do projeto é produzido pela *Business Expert, Designer de chumbo,* e a *Patrocinador* no *papel coordenador.* É aprovado pelo *Patrocinador,* referenciada por todos no projeto, e atualizado pela *Coordenador.* É produzido muito no início do projeto, antes do lançamento de plano de projeto grosseiro. É provável que o mapa projeto se transforma no plano do projeto como as datas são adicionadas, e as alterações são feitas diretamente no plano do projeto como as alterações na forma do projeto.

Um bom mapa projeto respostas em resumo: "Em que ordem que estamos entregando o que nós construímos ao lado Quais são as dependências entre os nossos lançamentos??"

Algumas equipas de projeto descartar o mapa projeto tão logo o *Plano de lançamento* é produzido, outros acham que é útil para reter. Pessoalmente

Aqui está um exemplo:

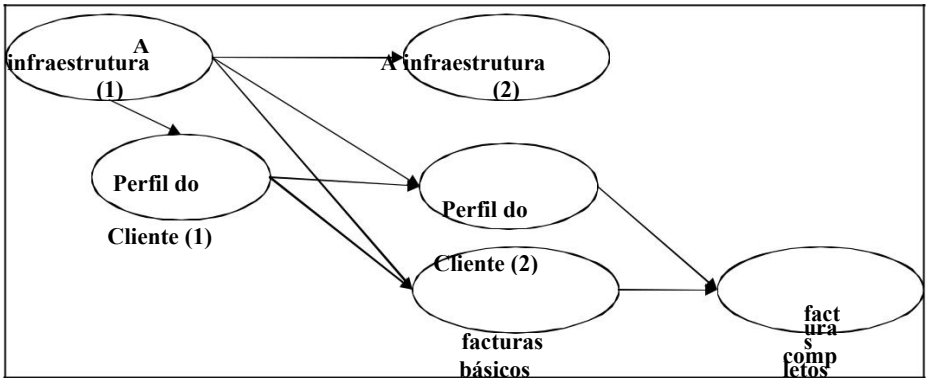


Figura 5-8

coordenador: **Plano de lançamento**

o *Plano de lançamento* é uma associação de datas com grandes etapas, que são geralmente extremidades de iteração e entregas. o *Mapa projeto* pode tornar-se a *Plano de lançamento* como datas são adicionados. o *Plano de lançamento* pode-se tornar-se diretamente o *Status do projeto (* o plano de lançamento NICS-NBO, abaixo, evoluiu para que o projeto de *Status do projeto).*

Chegando-se com datas requer entrada de todos. Normalmente, o primeiro *Plano de lançamento* é produzido pela *coordenador* e *Designer-chefe.* o *Business Expert* e *Patrocinador* verificar as prioridades do plano. O plano é reconstruída no início de

cada *Ciclo de entrega,* possivelmente, até mesmo cada iteração (lembre-se, existem várias entregas por projeto, e um ou mais iterações por *Ciclo de entrega).* Assim que a equipas está familiarizado com a atribuição de projeto, toda a equipas pode contribuir para a atualização do plano de projeto.

Aqui estão alguns exemplos de projetos reais, em formato gráfico e texto. *Comente sobre os formatos utilizados:* Muitos equipas leva começando com Crystal Clear e outras abordagens de desenvolvimento ágil sentir obrigado a colocar seu plano em um gráfico de Gantt. gráficos de Gantt obter rapidamente fora da data, são entediantes para atualizar, e são difíceis de ler. líderes de projeto e patrocinadores que se mudaram de gráficos de Gantt para diagramas de dependência e listas relatam que estes últimos são mais fáceis de ler e manter.

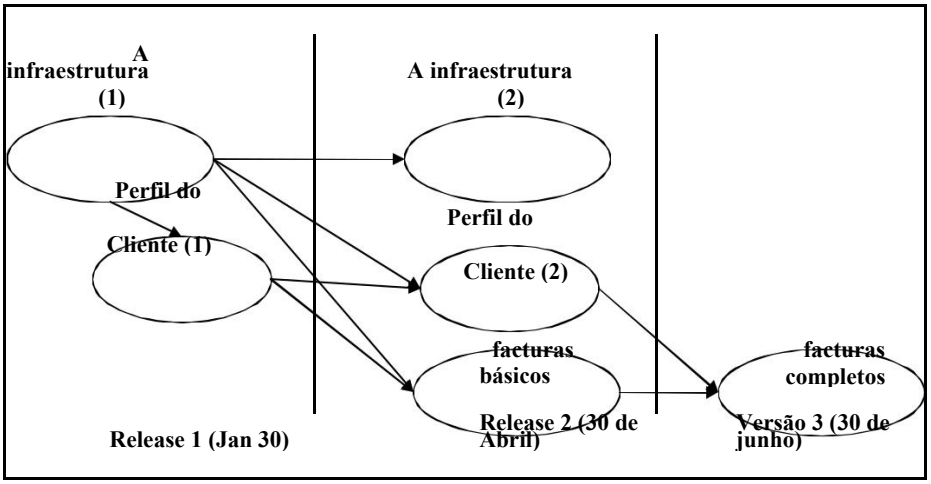


Figura 9/5

Release 1, Jan.30: função única. O comprador pode entrar um pedido que foi preenchido e assinado no papel, e gerar OP que podem ser impressos. Receptor pode marcar a entrega contra PO e pedido. Suporta dividida e fornecimentos parciais.

Release 2, 30 de abril: Primeiros relatórios (Pesquisas, coleta e estampas selecionadas relatórios) e funções primeiros segurança. Permite Solicitante para criar pedido, recolhe a aprovação do aprovador, notifica Comprador.

Versão 3, J une 30: fu completa n ction, com functio banco de dados completo ns.

Ator	Objetivo	Liberação
solicitante	*Iniciar uma solicitação*	1
	Alterar um pedido	1
Comprador	*pedido completo de ordenação*	1
	Iniciado PO com fornecedor	1
recebedor	*Cadastre entrega*	1
Qualquer	*Verifique sobre os pedidos*	1
autorizador	*autorizações de mudança*	2
aprovador	*pedido completo para a apresentação*	2
Comprador	*contatos mudança de fornecedor*	3
autorizador	*Validar a assinatura do aprovador*	3
solicitante	*Cancelar um pedido de*	4
	Mark pedido entregue	4
	Recusar bens entregues	4
Comprador	*Alerta de falha de entrega*	4

A Figura 5-10

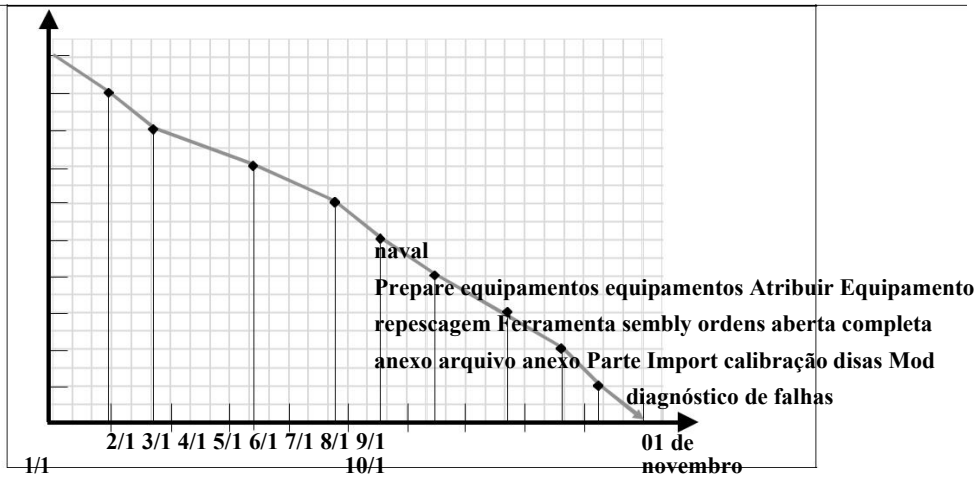
	Marco histórico	Planejado
I 1 A	única transação, simples faz uma viagem de ida e volta entre o computadores.	01 de fevereiro
I 2 tr	ansação de compra simples postado com sucesso de GUI através de base de dados.	31 março
I 3 R	elatórios de pedidos activos e work-flow através Comprador.	30 de junho
I To 4 d	o o fluxo de trabalho amostrados. Todos os relatórios são executados.	01 de setembro
I To 5 d	os os trabalhos de funcionalidade	30 de setembro
I 6 Ac	eitação do utilizador	10 de outubro
I D 7 e	sdobramento, desenvolvimento	20 de outubro

A Figura 5-11

Nome do recurso	Módulo	Valor	liberação #
1.1	Configuração gera comentários do item de linha ordem	M	1
1,2	Configrator UI Retrabalho: estilo assistente Verbose	H	1
2.1	PO gerado correctamente para a configuração	H	1
2.2	Criar / confirmar itens de fornecedores existir para skus	H	1
3.1	Encomendar Formulário avançado mostra mais detalhes	M	1
3.2	Ordem cumprida pelo custo PO	H	1
3,3	pedidos de repetição trabalha com configurações cegos	M	1
3,4	comentários de configuração são visíveis, não editável	eu	1
4.1	Base de Dados de mudança hierarquia	H	1
4,2	Estilo pode localizar gráficos de preços com base na seleção de cor H		1
4.3	Atribuir cores específicas do grupo de cor para Preço gráficos	H	1
1.3	Configuração gera preços específicos do cliente	H	2
1,4	Permitir a adição de uma configuração e continuar	eu	2
2,3	geração PO é automática	eu	2
3,5	Order mostra quadrados. Filmagem e rodando comprimento	eu	2
3,6	Vendedor itens que podem ser solicitadas mostrar detalhes adicionais	eu	2
3.7	tratamentos de janela impressos mostram renúncias	M	2
3.8	Permitir a cópia fim item de linha	H	2
4.1	estilo associado com Retail.net DCL	eu	2
4,11	grelha	eu	2
4,6	Atribuir cliente específico descontos de venda	H	2

Figura 5-12. (Graças a Jeff Patton e Tomax)

O seguinte é adaptado de um projeto de reengenharia para substituir um sistema de fluxo de trabalho. Cada nome é o nome de uma estação de fluxo de trabalho. Eles não usam um gráfico burn-down para o seu plano de projeto. Eu incluí-lo como um exemplo de como se poderia fazê-lo.


naval
Prepare equipamentos equipamentos Atribuir Equipamento
repescagem Ferramenta sembly ordens aberta completa
anexo arquivo anexo Parte Import calibração disas Mod
diagnóstico de falhas
1/1
2/1 3/1 4/1 5/1 6/1 7/1 8/1 9/1
10/1
01 de
novembro

A Figura 5-13

coordenador: **Status do projeto**

o *Status do projeto* é uma listagem do estado do projeto em relação ao *Solte Plano.* Muitas vezes, é exatamente isso com o status marcado. A informação de estado pode incluir a data actualmente prevista devido, ou a data de conclusão inicialmente previsto, data de conclusão prevista, e riscos previstos. Em um projeto, fomos convidados a também comparar os números de custo originais e projetados neste gráfico. O status do projeto geralmente deve levar menos de meia página; você pode usar a outra metade da página para a lista de riscos, fazendo um relatório do projeto completo em uma única folha de papel.

O status do projeto é mantido pelo *coordenador* em conversa com a equipas de desenvolvimento. Ele é usado pela *Patrocinador* eo gerente da comunidade de utilizadores para planejar seus esforços separados. Ele é atualizado a cada poucas semanas ou todos os meses.

Um bom status do projeto é breve, fácil de ler, e mostra a informação que os leitores precisam para avaliar o custo, data e trajetória de risco do projeto.

Aqui estão alguns exemplos. *Comentários sobre a estratégia e formatos utilizados:* As pessoas estão cada vez mais usando tabelas simples para relatar status. O segundo exemplo mostra uma das diferenças nas técnicas de desenvolvimento ágil: algo de importância negócios é entregue aproximadamente a cada semana, com um ciclo completo a cada mês. Ao completar uma conversão completa no primeiro mês, a equipas tem a chance de aprender o que está envolvido, aonde eles estão correndo em problemas, e ter bastante tempo para aprender a fazer melhor em ciclos subsequentes.

T h plano e projeto para o NICS-NBO p rojecto ser chegou a sua carta de status:

	Marco histórico	Notas entregu	es planejadas	
I 1 A t	ransação única e simples faz uma viagem de ida entre os computadores.	01 de fevereiro	03 de fevereiro	Completo
I 2 tra	nsação de compra simples postado com sucesso de GUI através de banco de dados.	31 março		Andamento. A incerteza sobre os componentes DCOM, ainda na curva de aprendizado. Experimentos em andamento.
I 3 Re	latórios de pedidos activos e work-flow através Comprador.	30 de junho		Começado. trabalho de design inicial.
I 4 Tod	fluxo de trabalho amostrados. Tudo relatórios executado.	01 de setembro		não foi iniciado
I 5 Tod o	os trabalhos de s funcionalidade	30 de setembro		não foi iniciado
I aceit	ação 6 Utilizador	10 de outubro		não foi iniciado
I 7 Im	plantação	30 de outubro		não foi iniciado

A Figura 5-14

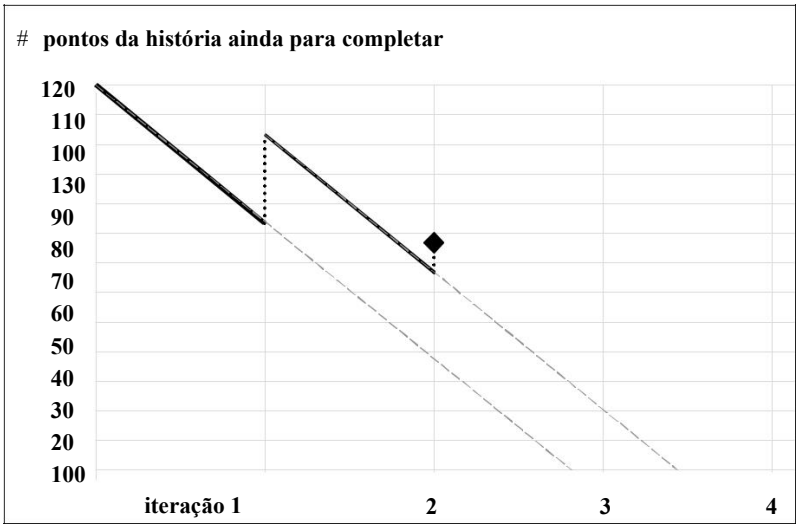
Aqui está uma folha de estado para um projeto envolvendo a conversão de 200 aplicações e suas bases de dados. Na preparação deste exemplo para publicação, eu só contaram os sistemas, aonde a empresa teve nomes de sistemas. As datas entre parênteses são o co previsto mpleti o datas n, os sem parênteses um r e compl real datas Etion.

Novo sistema de tabelas de código de Migração operando obras antigas tabelas convertido tabelas velhos removidos				
1	(Fev 1) 03 de fevereiro	(Fev 7) 10 de fevereiro	(Fev 14) 18 de fevereiro	(Mar 1) 01 de março
2	(Mar 1) 01 de março	(Mar 7)	(Mar 14)	(Abr 1)
3	(Abr 1)	(Abr 7)	(Abr 14)	(1 de Maio)
4	(1 de Maio)	(7 de maio)	(14 de Maio)	(1 de Junho)
5	(1 de Junho)	(7 de junho)	(14 de Junho)	(1 de Julho)

(Etc)

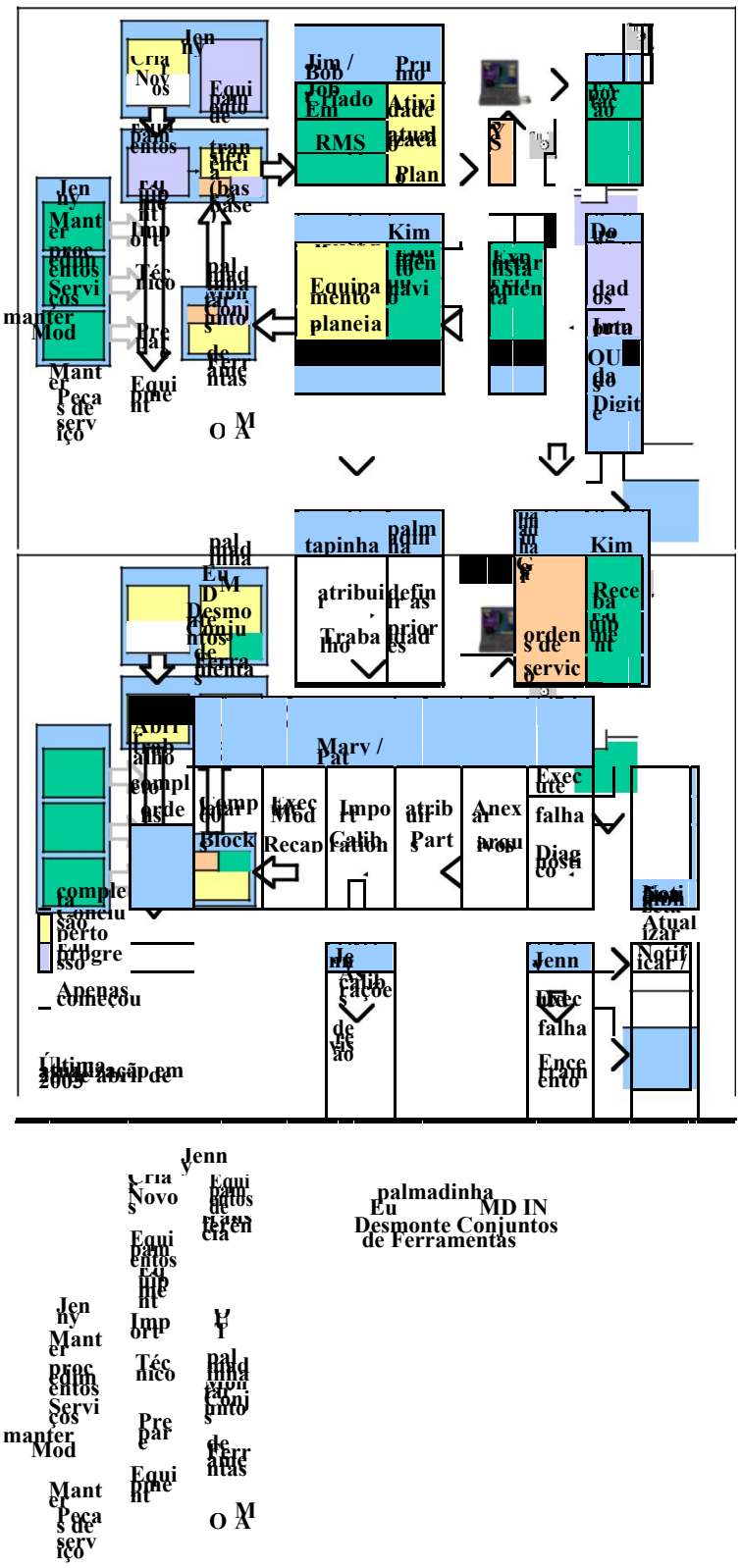
A Figura 5-15

Aqui é o status relatado em um *queimar* gráfico. Veja a técnica, *Queime gráficos,* para mais detalhes sobre este método.



A Figura 5-16

O seguinte é um gráfico que mostra a mudança de status ao longo de um período de dois meses. Branco significa não começou, (verde no original) meio-cinza significa concluída.



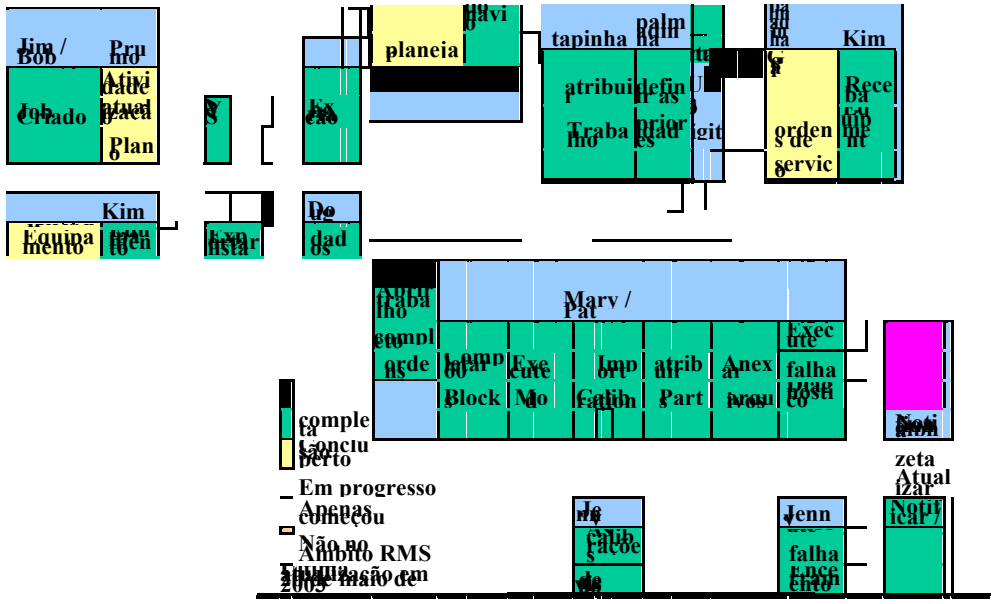


Figura 5-17. (Graças a Omar Alam)

coordenador: **Lista de Riscos**

A lista de riscos consiste dos principais riscos enfrentados pelo projeto, como significativa e provavelmente cada um é, os danos que pode causar, e que a prevenção ou resposta é possível. Eu gosto de controlar se eles têm crescido ou diminuído recentemente.

A lista de riscos é produzido e mantido pelo *coordenador* em conversa com a equipas, atualizado semanalmente ou mensalmente. Ele é usado em discussões com o patrocinador.

Criando a lista de riscos é trivial. A parte mais difícil é decidir o que fazer em caso da ocorrência do risco. Tal como acontece com a declaração de missão, a brevidade do produto de trabalho desmente o pensamento posto em construí-la. Há um sentimento muito gratificante de ver materializar um risco e perceber que um tem uma resposta pronta para isso.

A lista de riscos pode viver em um radiador de informações ou em uma planilha. Aqui está um exemplo do projeto NICS-NBO. Esta mais o status do projeto resultou em uma de uma página folha de resumo do projeto examinado em cada reunião do comité director.

Risco priorid	ade	resultado possível	Response probab	ilidade	Alterar Mile	stones afetado
1	Muito alto desempenho exige em uma nova tecnologia, não testada	bastantefalhasinesperada selevado em operação	do sistema	desempenho teste inicial	como antes	2-5
2	comunidade de utilizadores muda a sua ideia de requisitos na noite de	atrasos no meio de entrega		mostrar aos utilizadores o mais rápido possível	como antes cin	co
3	Precisa testar no chão de fábrica, bem como no desenvolvimento	mal-entendidos, atrasos na entrega	baixo	rotinas de teste claros	Menos	2-6
4	compatibilidade 2000	dados ruins no sistema provoca um funcionamento defeituoso	médio	teste Y2K especial	Menos	6
5	problemas de conversão de dados	Atraso na entrega	médio	criar estratégia de conversão	como antes	6-7

A Figura 5-18

coordenador: **Plano de iteração ‡ Estado iteração**

O plano de iteração de grão fino lista o que está a ser desenvolvido nesta iteração. Cada item pode ser algo que pode levar de um dia a duas semanas para se desenvolver, dependendo de quanto tempo a iteração é. O plano de iteração pode mostrar dependências entre itens de trabalho (se, por exemplo, foi produzido *Blitz Planning),* pode tratar todos os itens de trabalho como sendo independente (se, por exemplo, foi produzido pelo jogo de planificação do XP ou é simplesmente uma lista de características do produto), ou poderia ser uma marca de frases selecionadas nos casos de uso.

O plano de iteração é produzido pela *coordenador* trabalhando com a equipas de desenvolvimento. Ele é mantido até à data pelo *coordenador* da mesma forma, geralmente tornando-se o status de iteração diretamente.

Um plano de iteração boa lista todos os itens de trabalho a equipas tem que completar para que possam ser verificados off quando concluído, e para que o patrocinador e os desenvolvedores podem ver todos os itens de trabalho significativos. A granularidade varia com a duração da iteração e conhecimento da equipas de sua atribuição.

Aqui está um plano de iteração, cortesia de Thoughtworks corporação. Neste caso, uma história de utilizador XP está escrito em cada flipchart. Tarefas para cada história são escritos em notas pegajosas e anexado ao flipchart (as tarefas abaixo os cartazes foram tomadas fora do escopo da iteração).

A Figura 5-19 (Graças a Thoughtworks)

* * *

O status iteração lista o estado da iteração com relação ao plano. Muitas vezes, é escrito em um radiador de informações. Geralmente é apenas a iteração itens do plano com seu estado atual trabalho, geralmente marcados quer *começado* ou *feito,* com nada entre eles. Tenho visto alguns exigem uma terceira marca, *integrado,* para quando o item passa

testes de integração. Embora as tarefas são muitas vezes tão curto que não é realmente útil para marcar% concluída, o primeiro exemplo abaixo não mostra uma forma criativa de fazer isso.

O status iteração é criado pelo *coordenador* em conversa com a equipas de desenvolvimento. A atualização é, em princípio, feita por cada desenvolvedor, mas normalmente isso exige um esforço dirigido pelo *Coordenador.* O gráfico de estado serve para transmitir a todos a taxa de movimento através de lista de trabalho da iteração.

Um bom estado iteração é fácil para que todos possam ver e mostra de relance o que foi feito contra o que precisa ser feito.

Aqui é um status de iteração para as histórias de utilizador em cartazes, acima. Cada um foi copiado para uma nota, começou na parte inferior esquerda do gráfico, e mudou-se para a direita à medida que progredia, como a sua qualidade melhorou. Os espectadores poderia dizer o estado de cada história de utilizador por quão longe para a direita e quão alto cada história de pegajosa era. Esta foto foi tirada no final de uma iteração de um mês.

Figura 5-20. (Graças a Thoughtworks)

Este é um plano de iteração marcado para o status de iteração. As duas caixas à esquerda indicam "começou" e "feito" (há, é claro, não interessante estado intermediário). Os números à direita são estimativas tamanho relativo.

Neste gráfico, os cartões de índice vermelhos são os cartões de tarefa da sessão de planificação. Eles copiaram o texto no quadro branco para acompanhar iteração status. Aqui vemos a equipas usando iterações de uma semana, com 15 dias de trabalho disponíveis entre eles. Eles estão estimando que possam obter 9,5 unidades relativas de trabalho concluído esta semana.

A Figura 5-21 (Graças a Jeff Patton e Tomax)

Cada vez mais, muitas vezes vemos pessoas mover as notas em colunas para mostrar a mudança de status. As colunas são: *não foi iniciado, começado,* e *feito.*

Esta foto foi tirada em seu thrice- reunião semanal. Géry está se movendo marcador de um item do *começado* para *feito.*

A Figura 5-22 (Graças a G E ry Derbier.)

coordenador: **Visualizando Cronograma**

O cronograma de visualização é uma lista de quando, durante a iteração ou ciclo de entrega da *Embaixador Utilizador* ou outros utilizadores devem planejar para vir e ver o novo crescimento do sistema. Este produto de trabalho é necessária quando o tempo dos utilizadores não é tão fácil de obter, eles têm que viajar muito, ou há muitos deles. Não é necessário se o *Embaixador Utilizador* visita o projeto todos os dias ou todas as semanas.

O calendário de visualização é produzido pela *coordenador* em cooperação com a equipas de desenvolvimento no início de uma iteração. A equipas de desenvolvimento usa-lo como um lembrete de que a meta de conclusão do seu trabalho (milestones menores). É um documento vivo, o *coordenador* mudá-lo conforme necessário para equilibrar duas forças opostas: dando aos espectadores testadores como aviso muito antecedência possível e dando aos desenvolvedores mais tempo para reagir às mudanças em seu design. Pode viver em um radiador de informação ou no e-mail.

Como observado na secção *Reflexão sobre o processo,* e também discutido em " *Nós colocated e correu iterações de duas semanas - por que nós não ?,"* o cronograma de visão é mais importante para o resultado do projeto de seu tamanho modesto indicaria.

Este exemplo é a partir do sistema de rastreamento de Escoteiros:

Release 1: 01 de marco de 2002. sistema de entrada para permitir scoutmaster para inserir informações pessoais sobre olheiros e pessoal, incluindo nome, endereço, telefone, email, rank (quando aplicável), etc.

Vendo 1: 15 de fevereiro de 2002

As capturas de tela ou desenhos mock-up de tela

Vendo 2: 22 de fevereiro de 2002

bases de dados concluídas e telas GUI

Vendo 3: 01 de março de 2002

Conclusão do Release 1

A Figura 5-23

***Business Expert* e *Embaixador Utilizador:* lista ator-objetivo**

A lista ator-meta é um diagrama de tabela de duas colunas ou caso de uso. Na forma de duas colunas, os nomes das colunas à esquerda da pessoa, organização ou sistema de computador que irá conduzir o sistema, invocando as suas promessas de serviço. Os nomes das colunas direita que prometem serviço, que é o objetivo do ator em relação ao sistema, o que ele / ela / ele quer realizar usando o sistema. O objetivo, uma frase verbo suma, torna-se o nome do caso de uso descrevendo os detalhes da função.

A lista ator-objetivo é criada principalmente pela *Business Expert* e *Embaixador do utilizador,* e é revisto para a integralidade e priorização pela *Patrocinador.* Ele fica referenciado por todos. Ele é criado dentro ou imediatamente após o *fretamento projeto* atividade, e atualizado conforme necessário ao longo do projeto.

Uma boa lista ator-objetivo captura cada ator primário (aquele que vai conduzir o sistema) que o sistema deve reconhecer (deve mostrar um comportamento específico para), e para cada ator primário, cada grande objetivo desse ator no que diz respeito ao sistema. Metas são demonstrados ao nível "tarefa do utilizador", ou seja, transações completas de valor para o ator. (Meta níveis encontram-se descritos em detalhe em *Casos de Uso eficaz da escrita).* Uma lista ator-objetivo bem construído contém entradas que os *Patrocinador* e *Business Expert* sentem estão em um nível razoável e interessante de actividade (não muito detalhada, não muito alto nível), e eles podem rever a declarar que ela é completa, omite um objetivo, ou tem metas desnecessários.

A equipas do projeto pode decidir começar a trabalhar no projeto, mesmo quando o ator-objetivo é conhecido por ser incompleta.

A lista ator-meta é um índice para a funcionalidade total necessário do sistema. Ela serve tanto como uma declaração de requisitos de baixa precisão e também como um andaime para o status do projeto. Ele é usado para ajudar a construir tanto o plano do projeto de granulação grossa (veja o exemplo final) e o plano de iteração de granulação fina. Algumas pessoas usá-lo diretamente em relatar plano e status do projeto.

Quando usado para relatar o status do projeto, os nomes de casos de uso (metas) são úteis para os primeiros lançamentos. Mais tarde na vida de um sistema, as funções sendo solicitados tendem a ficar tão pequena que não valer a pena escrever casos de uso para. Eles tendem a ser *características* tais como, "hifenização", "exportação para o formato RSS," e assim por diante. Uma vez que esse momento seja atingido, a equipas muitas vezes muda para acompanhar o progresso de granulação fina de recursos, em vez de casos de uso. Por que não podemos apenas começar com as características, então? A primeira razão é que geralmente há muitos recursos. A equipas precisa de uma lista mais curta para trabalhar, para ver de relance o que eles vão conseguir. A segunda razão é que, no início do projeto, os utilizadores precisam ver o que o sistema vai fazer por eles no seu contexto de trabalho. Que é difícil de obter a partir de listas de recursos, mas é exatamente o que os casos de uso fornecem.

Eu incluir a lista ator-objetivo como um produto de trabalho separado do arquivo de requisitos (que segue), porque ele é criado mais cedo do que o resto do arquivo de requisitos, e muitas vezes ela é mantida separadamente.

Aqui está um exemplo:

Ator	Objetivo
solicitante	*Comprar algo iniciar uma solicitação*
	de alteração de um pedido de
	cancelamento de um pedido Mark
	pedido entregue
	Recusar bens entregues
autorizador	*autorizações de mudança*
Comprador	*contatos mudança de fornecedor*
aprovador	*pedido completo para a apresentação do pedido*
Comprador	*completo para encomendar Iniciado PO com*
	fornecedor
	Alerta de falha de entrega
autorizador	*Validar a assinatura do aprovador*
recebedor	*Cadastre entrega*
Qualquer	*Verifique sobre os pedidos*

A Figura 5-24

Business Expert: **requisitos Arquivo**

> **Em uma conversa sobre os requisitos de volta em 2000, Kent Beck fez uma de suas declarações provocativas, que "não existem coisas como *requisitos,* há apenas *desejos."* Como tantas vezes, eu encontrei seu pronunciamento a ser direito sobre a marca. Tente reler este e relacionados seções com o pensamento em mente que quando alguém diz: "O requisito (s)..." Eles estão realmente expressar algo mais próximo de um desejo, o que provavelmente é ajustável em circunstâncias adequadas.**
>
> **Particularmente com incrementais *Entrega* com o produto e feedback do processo, há uma abundância de oportunidades para a comunidade patrocinador e utilizador ao saber que eles realmente não *exigir* o que eles originalmente pediu, mas talvez outra coisa seria mais rentável ou mais útil, ou simplesmente adequado nas circunstâncias.**
>
> **Crystal Clear já é difícil o suficiente para adotar, sem me quebrar convenção padrão mais uma vez, e chamando esta seção "O arquivo desejos." Portanto, eu fico com Requisitos. No entanto, plantar a semente que eles estão mais perto de desejos, e ver se isso melhora a comunicação entre os patrocinadores, utilizadores e desenvolvedores sobre o que deve ser produzido.**

requisitos Crystal Clear não se destinam a ser tão completa e abrangente quanto a ser especificações claras para um programador do contrato externo não está familiarizado com o domínio (especificações de requisitos de escrita que explícita seria um desperdício de dinheiro no contexto Crystal Clear). O arquivo de requisitos é destinado a manter em requisitos lugar informações que poderiam ser esquecidos, como registros de decisões que a equipas precisa se lembrar, dando um contexto para essas decisões.

Um arquivo requisitos bom para um projeto Crystal Clear comunica com as pessoas no projeto com sua base de conhecimento específico e canais de comunicação. Quanto mais eles sabem e quanto mais eles podem aprender a falar com alguém por perto, o mais breve os requisitos pode ser.

O arquivo de requisitos é uma coleção de informações indicando

- **o que está a ser construída,**
- **que se destina a usá-lo,**
- **como ele fornece valor e**
- **Que constrangimentos grande afetar o design.**

O arquivo de requisitos pode ser um documento escrito, ou pode ser simplesmente arquivos espalhados pelo disco que, em conjunto, compreendem *os requisitos.* Em muitas organizações o conceito de requisitos como *documento* é tanto estrangeiros e desnecessário. O que é importante é que não existe um lugar aonde essas coisas são gravadas.

Há muitos formatos sensíveis que podem ser usados para o arquivo requisitos, e muitas estratégias de temporização em *quando* para preencher os detalhes.

- **Em alguns casos, como por um preço fixo, oferta fixe-o escopo, os requisitos devem ser bloqueados no início do projeto.**
- **Em outros casos, como por produtos retráctil, o desenvolvimento in-house, ou contratos de tempo e materiais, os requisitos podem ser evoluem ao longo do tempo ou ser criados de uma forma just-in-time.**

O Crystal Clear não legislar qualquer estratégia de formato ou timing. Normalmente, porém, as exigências evoluem ao longo do projeto, e a equipas precisa para actualizar e adaptar às mudanças no início de cada iteração. A equipas precisa para discutir periodicamente em seus workshops de reflexão como e quantas vezes os requisitos devem ser actualizados.

Qualidade em um documento de requisitos é relativo para a equipas. Um teste para *clareza* e *plenitude* é se um executivo da empresa ou outro especialista em negócios da mesma empresa se senta com o especialista em negócios que escreveu os requisitos, eles podem estar convencido de que há áreas são esquecidas, e dentro de cada área, eles entendem o que está escrito. *abrangência* significa que que todas as questões foram pensadas (isto é, naturalmente, mais difícil para testar).

Em geral, um arquivo de requisitos de amostra completo é muito longo para inserir aqui. Um está disponível online em *http://Alistair.Cockburn.us/crystal/article* Ele descreve seções típicas em um arquivo de requisitos completa, como

- **a declaração de missão,**
- **a lista ator-objetivo, seguido de casos de uso com anotações, especialmente os requisitos de desempenho,**
- **regras de negócios específicas para os casos de uso que podem ser necessários; referências a outras fontes de regras de negócios, requisitos legais, etc.,**
- **descrições de dados, formatos e regras de validação,**
- **requisitos de tecnologia,**
- **E / S protocolos e formatos para comunicações externas**
- **(Um glossário de termos de negócios é muitas vezes recomendado).**

É **improvável que um tipo Crystal Clear do projeto vai precisar de todos estes escritos para baixo, uma vez que eles têm uma estreita comunicação com os utilizadores e da patrocinadora. Como um exemplo, Kay Johansen contribui o seguinte *completo* Requisitos exemplo, com a nota introdutória:**

> **Alistair,**
>
> **Eu cavou ao redor e veio com dois exemplos de "requisitos" de um projeto que eu estava em torno de 1999. Ele provavelmente iria contar como um projeto Crystal Clear: dois programadores em uma sala (um dos programadores é também um especialista utilizador) um casal de outros especialistas de utilizadores no escritório próximo, além de acesso aos "clientes pagantes." Alguns testes automatizados. Solte a cada 2 meses para os clientes.**

Nós elaborou os requisitos em conjunto com os peritos do utilizador para os dois novos recursos que mais definidas a atualização do produto 2.61. O produto foi um sistema de contrato Small Business Accounting / inventário / serviço. Os requisitos tenho enviado para os clientes solicitando esses recursos, para a sua revisão. Os clientes ficaram satisfeitos, e os programadores trabalharam a partir deste documento.

contratos

História do utilizador

? **Criar um contrato de serviço com um saldo pré-pago que é debitado com base nos serviços prestados Para o consumidor.**

? **O contrato mantém um saldo de lucros não realizados, saldo mínimo e valor a ser cobrado.**

? **O contrato deve incluir percentagens de trabalho e peças com desconto para os serviços prestados sob contrato.**

? **Como chamadas de serviço forem apagados, os custos de trabalho e materiais (menos descontos) são deduzidos o equilíbrio não ganho do contrato.**

? **O cliente é cobrado o valor de faturamento quando o saldo de lucros não realizados contrato for inferior a o saldo mínimo.**

alterações de código

? **contratos**

? **Adicionar um *Bill por Serviços Prestados* caixa de seleção para indicar que o saldo do contrato é debitado com base nos serviços prestados.**

? **Adicionar um *Saldo mínimo* ao contrato para que o contrato pode ser cobrado quando o saldo atual cai abaixo do saldo mínimo.**

? **Adicionar um *Renovação Faturamento Quantidade* para permitir que o sistema de faturamento para renovar o contrato e reabastecer o saldo do contrato.**

? **Adicionar um *Trabalho Taxa de Desconto* para cálculos trabalhistas.**

? **Adicionar um *Materiais Taxa de Desconto* para peças e acessórios cálculos.**

? **Armazenar estas informações adicionais no banco de dados.**

? **faturamento contrato**

? **O sistema de faturamento contrato precisa verificar se há contratos que são nossos contratos (Serviços Prestados) que têm ido abaixo do seu saldo mínimo. Ela irá então criar o faturamento do contrato com base no faturamento Valor Renovação.**

? **Billings que são negativas somará o valor absoluto do valor negativo com o Renovação Faturamento Valor para trazer a conta até o mínimo positivo.**

? **Console de despacho**

? **Quando uma chamada é liberado para os nossos contratos (por serviços prestados), todos os custos de trabalho e materiais**
será descontado pela *Trabalho Taxa de Desconto* e a *Materiais Taxa de Desconto* para o contrato, respectivamente.

? A chamada apuradas debitará o equilíbrio não ganho do contrato e a chamada irá mostrar uma saldo zero devido. Se os serviços exceder o saldo atual no contrato, o saldo do contrato vai negativo e irá faturar para o excesso de custos na próxima renovação.

? A chamada apuradas debitará o equilíbrio não ganhos na Conta GL e creditar a receita conta. Ele vai usar as mesmas contas especificadas para a acumulação transferências de equilíbrio e de receitas a apropriar.

Faturamento bloco Tempo

História do utilizador

?Criar uma ordem de venda para 10 horas de treinamento em US $ 100 / hr (sobrecarga de US $ 50 / hr) e 5 conjuntos de documentação em US $ 50 / e a (custo de US $ 25 / e a). Bill isso como cumprida.

? Facturar imediatamente por 3 conjuntos de documentação e 5 horas de formação.

? O cliente paga US $ 650 para este factura.

?Cumprir um conjunto de documentação e 4 horas de formação.

?Examine o status da ordem de vendas - vista quantidades encomendadas, cumpridas, e cobradas em ambos quantidade de dólar e de itens quantidades.

? Cumprir os restantes 4 conjuntos de documentação e 6 horas de formação.

? Fatura para os 2 conjuntos de documentação e horas 5 formação que ainda não foram faturados.

? O cliente paga US $ 600 para essa fatura.

? A ordem de venda foi concluída.

alterações de código

? Para criar uma ordem de venda que você usaria o mesmo método que existe agora. Horas de Formação teriam de ser configurado como um ICItem não-abastecido. O Código de vendas no item irá determinar aonde a receita não auferida vai.

? Você pode abrir a ordem de venda e visualizar o status atual de todos os itens de linha vendidos. este informação está disponível tanto em quantidade de dólar e quantidade. O botão de comando "Bill" permitirá que você gerar uma fatura para isso, ou uma parte deste, Ordem de Vendas. A ação de facturação irá debitar a conta A / R e Receita a apropriar de crédito.

? Quando o cliente paga a fatura a funcionalidade existente será usado com os ajustamentos necessários ao Caixa e contas de A / R.

? A ordem de venda pode ser incrementalmente cumprida abrindo a ordem de venda e usando o "Cumprir Itens" e "Cumprir Serviços" botões. Cumprimento fará os ajustamentos necessários ao apropriar Receita, Receita, Custo dos Produtos Vendidos, inventário e contas de custos aplicados.

? Um novo relatório será criado, que lhe permite ver o estado actual da Ordem de Vendas incluindo todos os itens de linha com quantidades encomendadas, cumpridas e faturados. Esta informação estará disponível em ambos os valores em dólares e quantidades.

? Se a Ordem total de vendas ainda não foi facturado no momento do cumprimento final, uma factura pode ser gerada naquele momento. O processo de factura irá debitar a conta A / R e de crédito a conta apropriar Receita.

?O cliente pode pagar essa factura final utilizando a funcionalidade existente. A Figura 5-25 (Graças a Kay Johanssen)

***Business Expert* e *Embaixador Utilizador:* Casos de uso**

Um caso de uso é uma coleção de texto de cenários que descrevem como um ator externo alcança algo de valor usando o sistema. Ela começa com um cenário de sucesso em que o externo (*primária)* ator pede algo do sistema. Esse cenário descreve as ações e interações entre o sistema e vários outros atores em satisfazer o pedido do ator primário. Depois disso, ele descreve como o sistema se comporta quando as coisas dão errado, possivelmente, não para satisfazer o pedido do ator primário. Os casos de uso descreve o que o sistema deve *Faz;* eles não capturar UI design, requisitos de desempenho, definições de interface, ou definições de dados.

Os casos de uso são criados conjuntamente por um *Designer-Programmer,* uma *Business Expert,* e um *Embaixador utilizador.* Responsabilidade pode mover-se entre esses três papéis. Eu escrevo que o *Business Expert* é responsável por eles, mas na minha experiência, não importa muito qual deles é primário, enquanto os outros são revisores e fornecedores de conhecimento. Ele não causar problemas se um deles está ausente. Não-programadores tendem a escrever livremente ou de forma ambígua e não considerar plenamente as consequências e alternativas. Os programadores são geralmente boa para aqueles, mas tendem a ficar muito detalhada, comece descrevendo seu projeto pretendido, e obter regras de negócio errado. Trabalhar e revisar juntos, eles podem cobrir erros uns dos outros.

Um caso de uso é escrito frequentemente em duas fases, o cenário de sucesso numa primeira fase, os cenários falhas mais tarde, quando a equipas precisa se aprofunda nos requisitos, tanto para estimar a complexidade do trabalho, ou para trabalhar no projeto. Em algum momento, os escritores ou a tam pode declarar que os casos de uso são "completa" para a iteração, o que significa que há mais casos de uso serão considerados para a iteração, e que há mais detalhes ou extensões adicionados a eles

Um bom caso de uso é fácil de ler, identifica as responsabilidades do sistema, e considera os casos de falha mais interessantes. O conjunto completo de casos de uso abrange todos os objetivos os principais atores têm com relação ao sistema, incluindo a cobertura de todos os eventos externos que aciona o sistema. Um bom caso de uso Crystal Clear é diferente do que um bom caso de uso em um projeto maior, distribuído ou crítica de vida. Porque a equipas Crystal Clear senta juntos ou muito perto e pode obter esclarecimentos sobre o conteúdo facilmente, eles podem escrever em um breve, estilo mais casual.

A maioria das pessoas ficam muito extravagante sobre o formato de caso de uso, tornando-os muito detalhada em vez de comunicativo. Para projetos Crystal Clear, eu sugiro começar com o *parágrafo Segundo* Formato. Este é constituído por um título, uma descrição do sucesso no primeiro parágrafo, e uma descrição de falhas e recuperação no segundo. Uma vez que a equipa tem praticado com a forma de dois parágrafos, eles podem investigar outros formatos de casos de uso (veja *Casos de Uso eficaz da escrita).* Por exemplo, apesar de eu entender a forma de dois parágrafos, eu costumo achar que é mais

fácil e mais legível para escrever com passos numerados. Eu sempre escrevo nível meta do caso de uso, o nome do sistema que está sendo projetada, e as partes interessadas do sistema e os seus interesses em caso de uso. Esta informação adicional

comunica muito para os leitores sem adicionar um monte de tempo para a escrita. Experimente a forma de dois parágrafos, em primeiro lugar, porém, antes de adicionar qualquer outra coisa.

Aqui está um exemplo de um caso de uso de metas de nível de utilizador no de dois parágrafos do sistema BSA:

Ver o progresso de um Scouts' (nível de objetivo do utilizador)

o *chefe de escoteiros* procura e seleciona um batedor individual. O sistema mostra as informações sobre o posto o batedor está trabalhando atualmente.

Se o batedor não está no sistema, o scoutmaster pode Adicionar um novo Scout.

A Figura 5-26

Aqui está um exemplo de um uso resumo, dois parágrafos do sistema PRTS. (Nota: Uma frase sublinhada representa um hiperlink para outro casos de uso.)

Comprar algo (nível de resumo)

o *solicitante* inicia um pedido e envia-o para ela ou o seu *Aprovador.* o *aprovador* verifica que não há dinheiro no orçamento, verificar o preço das mercadorias, conclui a solicitação para a apresentação, e envia para o *Comprador.* o *Comprador* verifica o conteúdo de armazenamento, encontrar melhor fornecedor de bens. o *autorizador* Valida *aprovador de* assinatura . o *Comprador* então completa o pedido de ordenação , inicia um PO com o *Fornecedor.* o *Fornecedor* fornece bens para *receber,* recebe recibo de entrega (fora do escopo de sistema em design). o *recebedor* entrega de registros , Enviar mercadorias para *solicitador,* quem pedido marcas como entregue .

A qualquer momento antes de receber bens, *solicitante* pode alterar ou cancelar o pedido . Cancelamento ele remove-lo a partir de qualquer tratamento activo. (Apagar do sistema?) Reduzir o preço deixa intacta no processo. Elevar o preço envia de volta para *Aprovador.*

A Figura 5-27

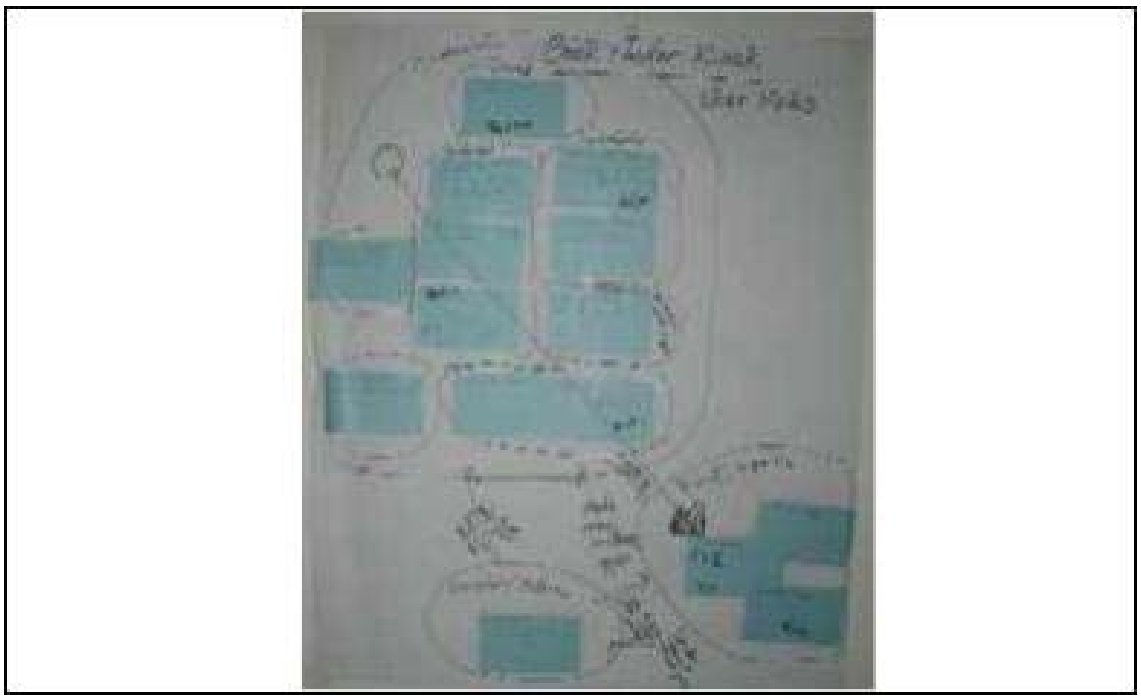
Embaixador Utilizador: **Modelo função do utilizador**

Um modelo é um mapa bidimensional mostrando funções de utilizador e seus relacionamentos. Ele destaca a *focal* papéis, as que o sistema deve satisfazer a maioria (ver *Interaction Design Essencial* na página 98). A função de utilizador é um nome que representa um objetivo de alto nível que o utilizador do sistema pode entrar; papéis semelhantes agrupar juntos e papéis diferentes aparecem mais distantes. linhas de relação conectar papéis ou clusters.

O modelo é criado conjuntamente por um *Designer-Programmer,* uma *Business Expert,* e um *Embaixador utilizador.* Embora a responsabilidade pode mover-se entre esses três papéis, eu anexá-lo ao *Embaixador Utilizador* para indicar a importância dessa pessoa no processo.

O modelo é usado

- **de priorizar o trabalho quando os papéis, o valor do negócio, estado focal e frequência de utilização são questões que afetam a prioridade de desenvolvimento de recurso,**
- **ao escrever casos de uso: as colaborações entre os papéis fornecer casos de uso com atores ou dependências adicionais,**
- **para ajudar a tomar decisões específicas sobre user-interface e design de interação,**
- **em testes: o testador assume as metas e os atributos de uma função de utilizador. Uma boa nomes de modelo papel todos os papéis que irão utilizar o sistema, revela rapidamente os papéis focais e relações-chave, e serve como um lembrete para a equipas como para que seu software estará servindo.**



A Figura 5-28 (Graças a Jeff Patton)

Designer-programadores: **Rascunhos de tela, sistema de arquitetura, código-fonte, modelo de domínio comum, design esboços e notas**

A pergunta sempre surge quanto ao que documentação é necessária. Em Crystal Clear a resposta é:

Seja qual for o patrocinador e a equipas decidir.

Há quatro razões para esta resposta:

- **O sistema estará sempre mudando, tanto em função e em design. O tempo, esforço e recursos colocados em documentar a subtrair projeto daqueles fazendo progresso em direção a entrega do sistema. A documentação será sempre um pouco fora de data e, portanto, incorreto.**

- **Ao mesmo tempo, os designers precisam deixar um rastro atrás, para correr as suas próprias memórias alguns meses para baixo da linha, e levar outros designers para entender o que eles têm feito.**

- **O acima estão em conflito, e a resolução correta deles depende do jogo de desenvolvimento de software que está sendo jogado neste projeto particular. Apenas os patrocinadores e a equipas pode adivinhar o que trilha é o ideal para deixar para trás, equilibrando a necessidade de progresso e a necessidade de pistas. O que é certo é que eu não posso, neste livro, tomar essa decisão para todas as pequenas equipas em toda parte.**

- **Finalmente, as tecnologias mudam. A forma correta de documentação muda um pouco com as tecnologias que estão sendo utilizadas. sistemas de banco de dados, sistemas de mainframe, sistemas científicos, sistemas orientados a objetos, sistemas de tempo real, todos têm diferentes formatos de documentação apropriada. Estas coisas mudam muitas vezes o suficiente para que, mais uma vez, não é para mim neste livro para legislar o que é correto para a equipas; só a equipas sabe.**

projetos Crystal Clear operar com dois horizontes de tempo com relação a documentação: recebendo o software escrito (o menos tempo gasto documentando o melhor), e entregando o software para um grupo diferente (mais rica a documentação, melhor). Pode-se pensar do primeiro conjunto de ações gananciosas (reunião o primeiro gol no jogo cooperativo), e o segundo como ações de investimento (criação de para o próximo jogo). Os dois estão em conflito uns com os outros, então a equipas inevitavelmente desempenha um jogo de malabarismo com a documentação.

Idealmente, a equipas produz a documentação de arquivo o mais tarde possível, o mais rápido e mais barato possível. Isso é permitido em Crystal Clear.

No entanto, "nenhum" não é uma resposta válida. Isso só seria uma resposta válida, se é sabido que ninguém vai manter ou ampliar o sistema mais tarde. Tal projeto é improvável que precisa mesmo Crystal Clear para seu conjunto de regras.

Assim, as equipas usando as regras de Extreme Programming dentro de um projeto Crystal Clear deve discutir com seus patrocinadores que formas de documentação para deixar para trás. Eles podem usar qualquer jogo de planificação do XP (Beck, 2002) ou o *Planificação Blitz* técnica para integrar essa necessidade de documentação para o trabalho do projeto em curso. Esta é a única além de XP necessária para ser uma implementação válida de Crystal Clear.

Além de discutir *o que* de documento, a equipas precisa para discutir *quão* documentá-lo, e *quando.* o *quando* parte

é fácil de entender, embora não tão fácil de escolher - o mais tarde eles deixá-lo, o mais up-to-date será e quanto menos ele terá de ser alterado. No entanto, o mais tarde eles deixá-lo, o mais provável é que eles não estão a fazê-lo em tudo. Há inevitavelmente brinkmanship envolvido em fazer a escolha ideal.

A equipas deve ser criativo sobre *quão* para documentar seus projetos. documentação digitado-in, em suporte papel é um dos mais caros, demorados e formas comunicativas menos disponíveis (não importa que é tradicionalmente o mais solicitado).

- pequenas equipas eficazes completar os seus documentos digitados-in com fotografias de seus quadros, cartazes e outros radiadores de informação.
- Não há nenhuma razão para que um diagrama desenhado à mão não podem ser digitalizados e armazenados on-line como
  uma imagem, incluído no HTML ou outros documentos.
- guardanapos de papel acontecem ser meu meio documentação favorito. Eles podem ser afixado na parede, fotografada ou digitalizada.
- A fita de vídeo da equipas podem um dos seus criadores, explicando e discutindo uma seção do projeto do sistema com um designer de *lado de fora* O time. Esta discussão deve ser realizada para 10-15 minutos por tópico, como descrito em *Desenvolvimento Ágil de Software.*

equipas inteligentes são susceptíveis de vir com suas próprias formas baratas e eficazes alternativas de documentação.

Os pontos a serem observados são de que

- *alguns* documentação é necessária,
- a equipas está em melhor posição para decidir o que, como e quando,
- a equipas em conjunto com o patrocinador estão em melhor posição para decidir o quanto é apropriado para este projeto específico. Finalmente, o desenvolvimento de software não é apenas um jogo de *comunicação,* mas também um jogo de *invenção.* Alguns produtos de trabalho facilitar a criação de novas ideias, ajudar as pessoas a fazer novos movimentos no jogo. -Papel baseado prototipagem (Snyder 2003) é um exemplo, como são os cartões de CRC (CunninghamURLcrc, CockburnURLcrc) e os diagramas de instância em UML (FowlerUMLD).

produtos de trabalho que aumentam a invenção não são arquivados, e, como consequência, tendem a ignorar a sua importância em um projeto. Para ajudar a corrigir esse desequilíbrio, eu chamo

esboços de tela como um produto de trabalho específico. Estas ajudar a equipas a trabalhar a interface de utilizador

projetar de forma colaborativa com os utilizadores. Eles são, produtos de trabalho úteis específicos com vida útil curta.

Nas seções a seguir, descrevo produtos de três de trabalho que quase certamente precisam ser produzidos - arquitetura do sistema, modelo de domínio comum, e rascunhos de tela - dentro da categoria maior, "esboços e notas, conforme necessário."

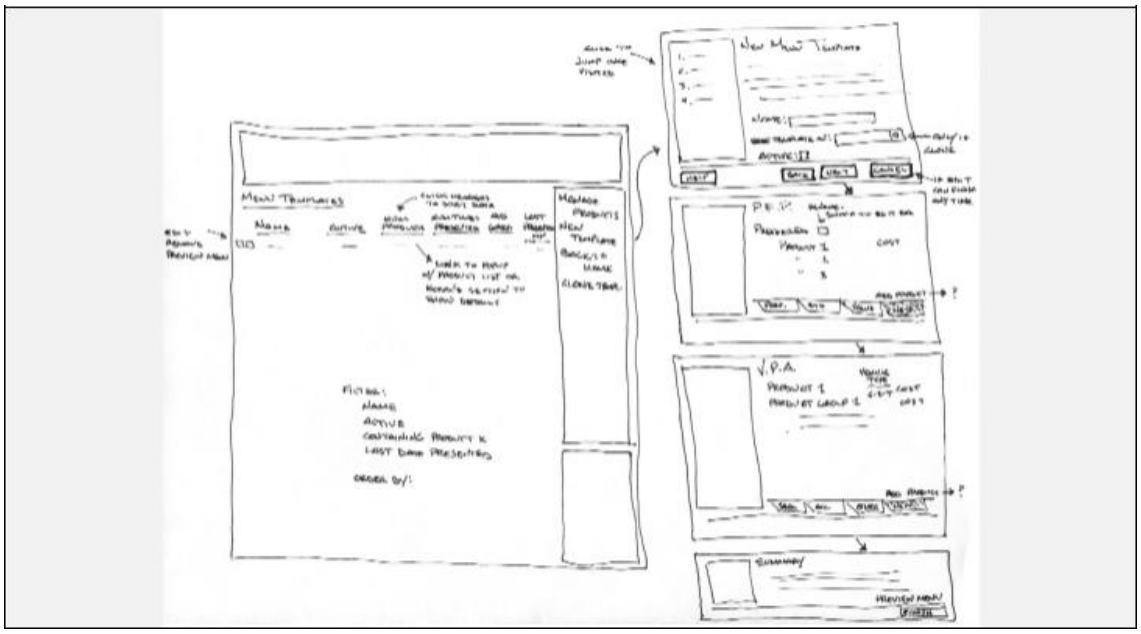
Designer-Programmer: **Rascunhos de tela**

esboços de tela são interpretações de baixo custo do caminho telas vai olhar, usado para explorar as maneiras que os utilizadores possam interagir com o sistema, e inventar maneiras melhores para eles para obter seus objetivos realizado. Muitas vezes eles são apenas desenhos em papel (Constantine ???, Snyder ???). A equipas pode preferir software de prototipagem tela.

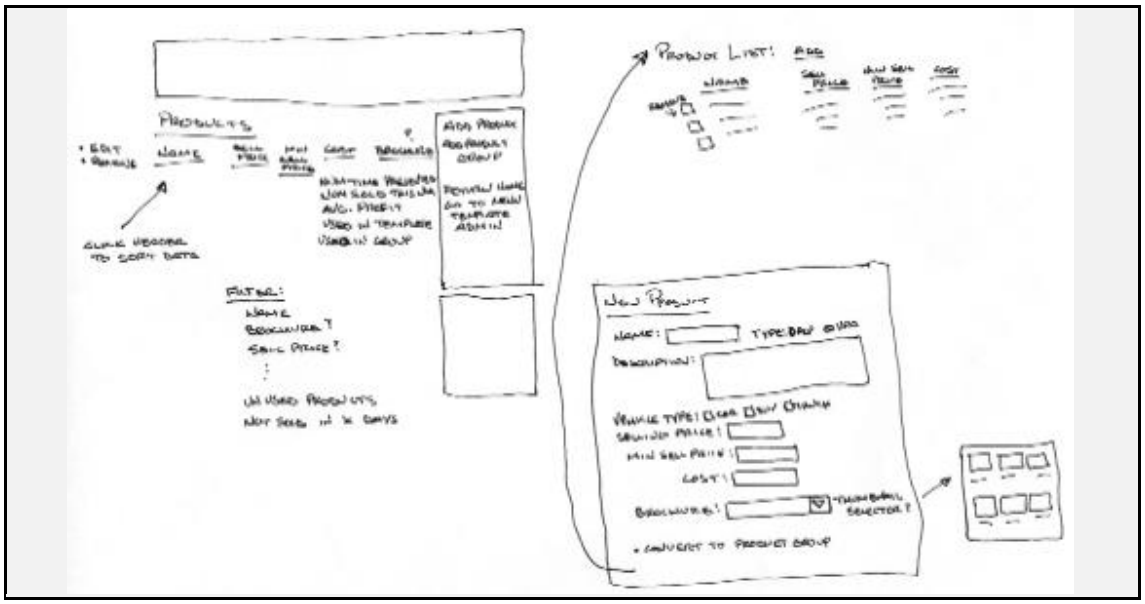
esboços de tela são criadas principalmente por qualquer *Designer-Programmer* é projetar a interface do utilizador, em conjunto com o *Embaixador utilizador.* Eles podem pendurar na parede durante a iteração, ou eles podem ficar convertido em protótipos de software ou código real e descartados. Eu mencioná-los aqui porque, apesar de serem produtos de trabalho transitórios, eles são imensamente útil e muitas vezes esquecido em projetos.

Um bom conjunto de esboços de tela é barata de construir (portanto, feita a partir de papel e marcadores ou de software especial encenação) e para o fácil *Embaixador Utilizador* e outros utilizadores para percorrer no que diz respeito aos casos de uso e cenários de uso (veja a técnica para walk-throughs em *Interaction Design Essencial).*

Aqui estão alguns exemplos de esboços de tela.



A Figura 5-29 (graças a Nate Jones)



A Figura 5-30 (graças a Nate Jones)

Designer-chefe: **Arquitetura do sistema**

A arquitetura do sistema descrição é texto e / ou desenhos mostrando os principais componentes e interfaces do sistema. Há muita controvérsia sobre o que constitui um *arquitetura,* quando deve ser feito e como deve documentado, todos os quais estão fora do escopo deste livro para decidir. Não tem necessidade de ser alguma descrição do projeto principal do sistema para unificar os membros da equipas trabalhar e educar designers de follow-on do sistema. Em projetos maiores, pode ocorrer duas arquiteturas: a arquitetura técnica (infra-estrutura) e a arquitetura de domínio. *Domain Driven Design (* Evans 2003) tem exemplos de arquiteturas de domínio.

A descrição da arquitetura do sistema é criado pelo *Designer-chefe,* geralmente bastante no início da primeira iteração. A arquitetura provavelmente irá evoluir, especialmente se o *caminhando de esqueleto* e *rearquitetura incremental* estratégias são utilizadas. Isto significa que a arquitetura documento terá de ser actualizado durante cada iteração.

Uma boa arquitetura de sistema fala na linguagem da tecnologia a ser utilizada eo *Designer-programadores.* Pode consistir em um memorando técnico com texto descritivo e desenhos, ou apenas o texto ou desenhos apenas. Ele mostra ou descreve os principais interfaces e os caminhos de interacção.

Aqui são exemplos de estilos muito diferentes de documentação de arquitetura:

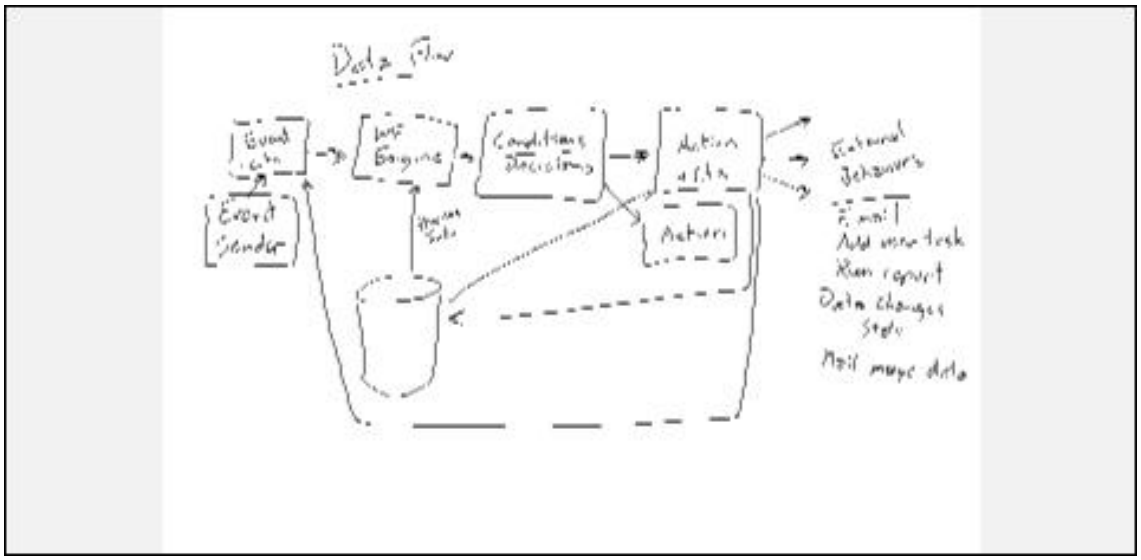


Figura 5-31. Projeto capturado por um dispositivo quadro Mimeo (graças a Darin Cummins)

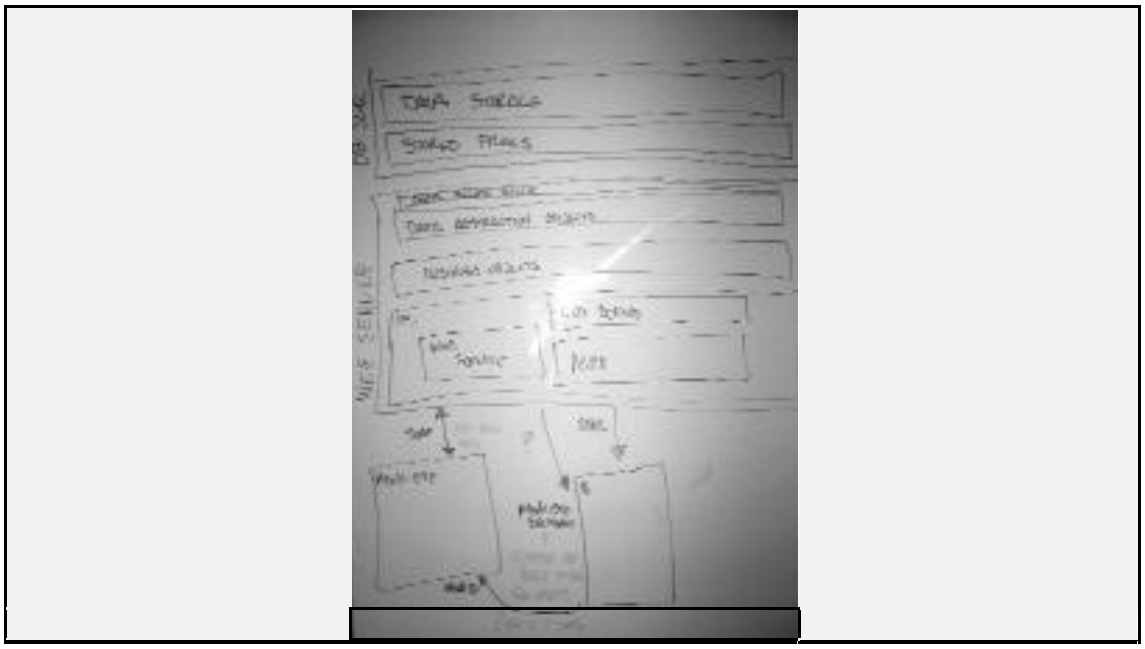
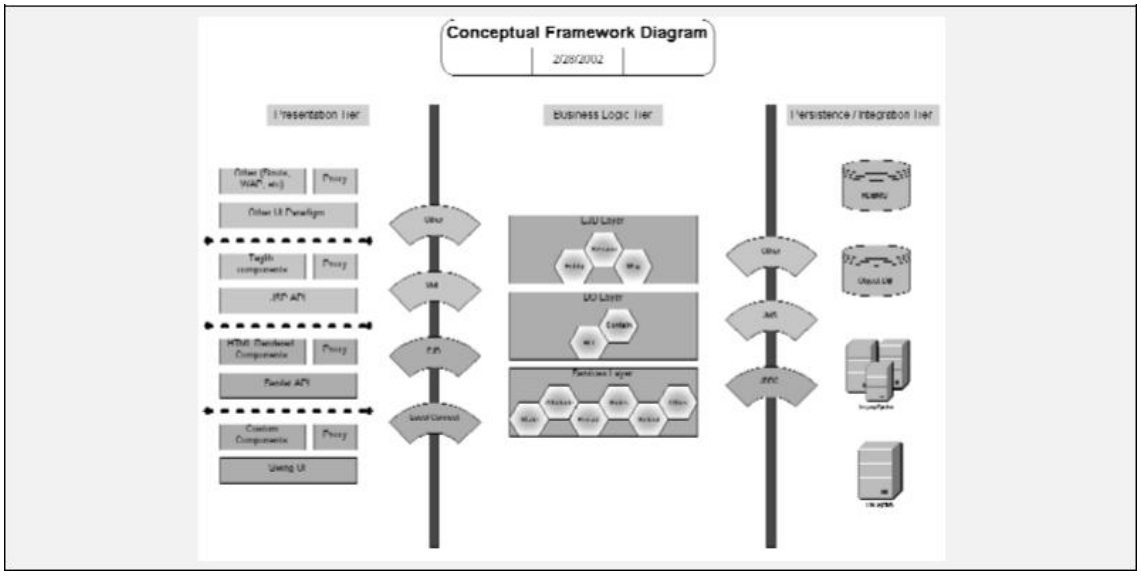
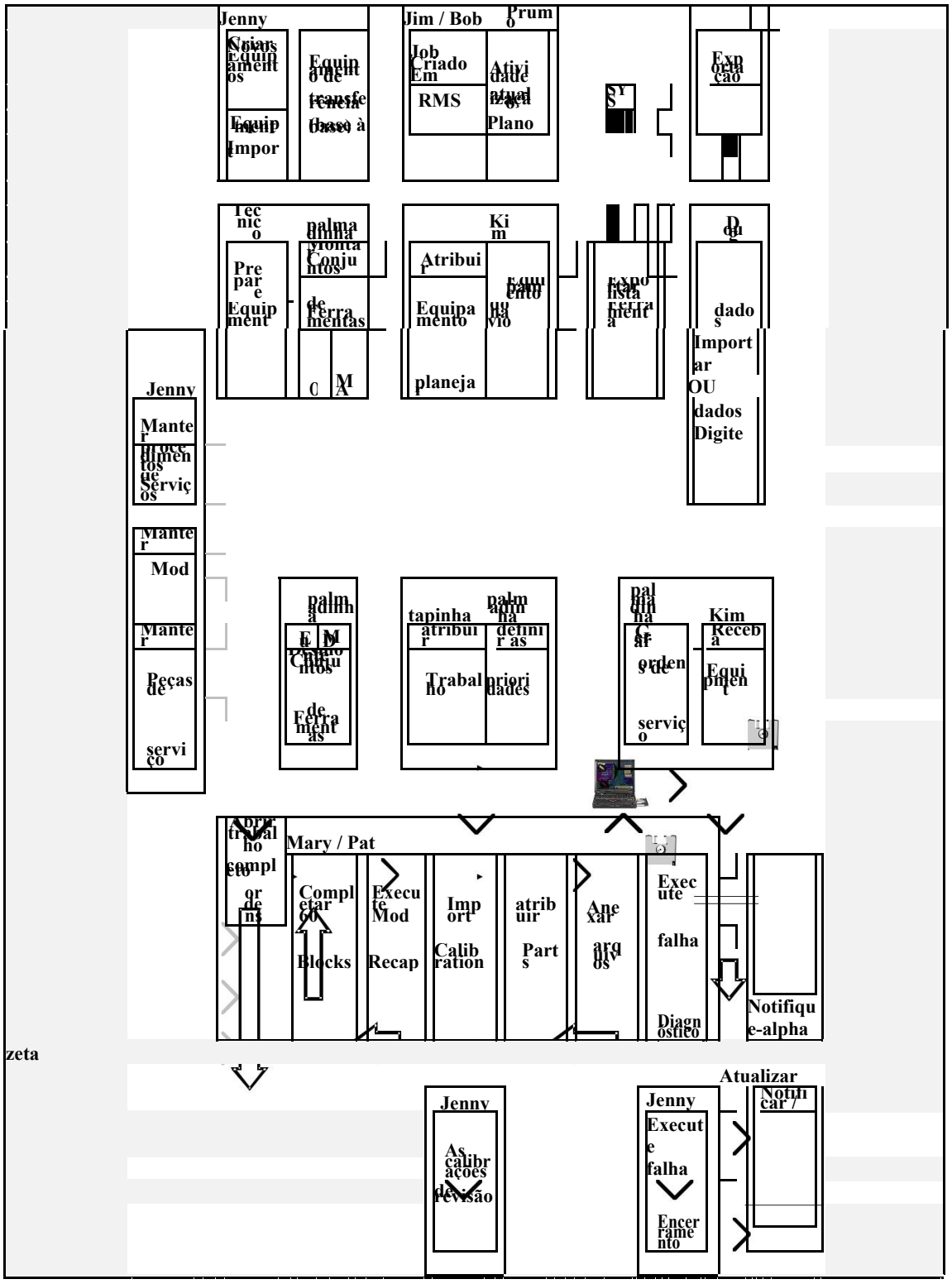


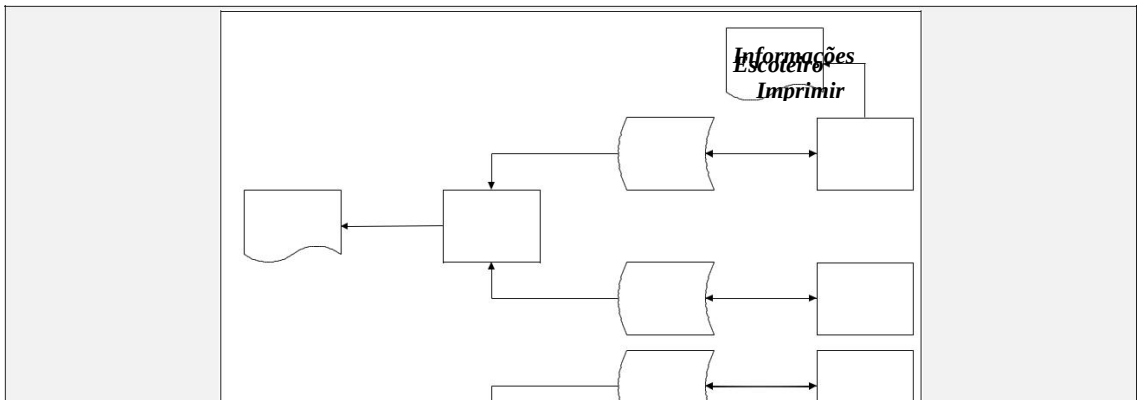
Figura 5-32. Arquitetura mantidos em um quadro branco (graças a Nate Jones)



A Figura 5-33 Arquitetura arrastado para uma ferramenta de desenho. (Graças a Jonathan House)



A Figura 5-34 arquitetura de fluxo de trabalho. (Graças a Omar Alam)



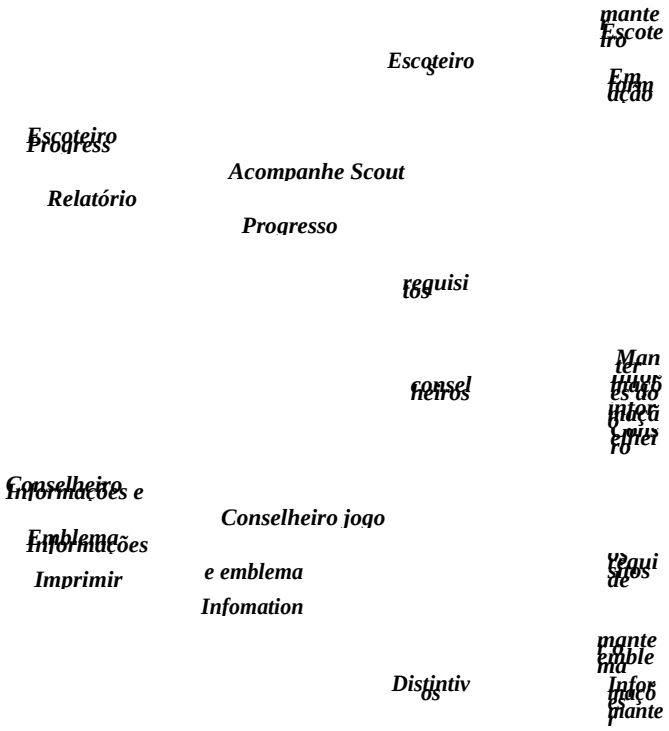


Figura 5-35. Arquitetura de um sistema não-OO

Designer-Programmer: **Modelo de Domínio comum**

O modelo do domínio comum é um desenho tipicamente um, ou um esquema de banco ou um diagrama de classe, mostrando as principais entidades ou classes usadas no sistema. Algumas equipas ainda preferem o texto diretamente para descrever o seu modelo de domínio.

O modelo de domínio comum é criado pelo *Designer-programadores* como eles entendem o domínio e incorporar quantidades crescentes de que em seu projeto. Ele é analisada pela *Business Expert,* que pode ajudar a construí-lo em primeiro lugar.

Existem várias estratégias de temporização para a construção do modelo de domínio. Em alguns casos, há uma atividade de domínio de modelagem intensa no início do projeto, o que resulta em projeto nº 1 do modelo de domínio comum. O modelo fica revistos e ajustados de forma contínua ao longo do projeto depois disso. Este é o modo de trabalho descrito *Projetos sobreviver Object-Oriented (* pp. ??). A outra estratégia é uma estratégia em tempo just-in-, em que o modelo é construído de forma incremental e minimamente, com pouco ou nenhum olhar em frente a necessidades futuras. O Crystal Clear não legislar qual dessas estratégias é o preferido. Isso é com a equipas para discutir e decidir.

Há uma razão pela qual eu chamo este produto de trabalho do "comum" modelo de domínio, ao contrário do modelo de "análise", o modelo de "design" ou apenas modelo de "domínio". Tendo vários modelos cria dois perigos para o projeto. O primeiro é a manutenção dupla: você tem que voltar a sincronizar cada modelo assim que quer mudanças. Normalmente, isso não é feito, então os modelos simplesmente obter fora de sincronia. A outra é que há uma tendência a pensar que o modelo de análise é magicamente "verdadeiro" de alguma forma, e o modelo de design precisa de justificativa para cada diferença. Na realidade, há muitos "verdadeiros" modelos do domínio. A implementação final deve conter um que é correto e um bom projeto com relação a recursos de manutenção, desempenho e do sistema. A, modelo de análise inicial pode estar correta, mas é bastante provável que seja fraco no que diz respeito aos critérios de projeto. Como resultado, é uma estratégia melhor para manter apenas um modelo do domínio, e evoluir lo para que ele é ao mesmo tempo um modelo de domínio válido e um design forte. Este tópico é discutido em detalhe no *Projetos sobreviver Object-Oriented (* Cockburn 1998). Eric Evans refere-se ao modelo de domínio comum evoluindo com seu padrão, *Idioma ubiquitous*

(Evans 2003).

Decidimos em projeto Winifred que o conjunto de classes para colocar no modelo de domínio comum devem ser os "elementos públicos de classes persistentes." Eu encontrei este para ser um guia útil. O modelo que resulta é muito o que um analista chamaria um "modelo de análise", o que significa que ele contém apenas os itens que devem estar na *Business Expert do (* e provavelmente a *Embaixador do Utilizador)* vocabulário e esfera de cuidar. Ao mesmo tempo, nomes de classes e relações que realmente existem na implementação - é um "modelo de implementação." Ter um modelo de análise que é um extracto do modelo de implementação mantém a equipas de manutenção de dois modelos, e aumenta a probabilidade de que as entidades e classes colocados na implementação, na verdade, são termos de negócios significativos.

Um bom modelo de domínio comum é tanto compreensível para o *Business Expert* e corrente com a implementação.

Aqui estão alguns exemplos.

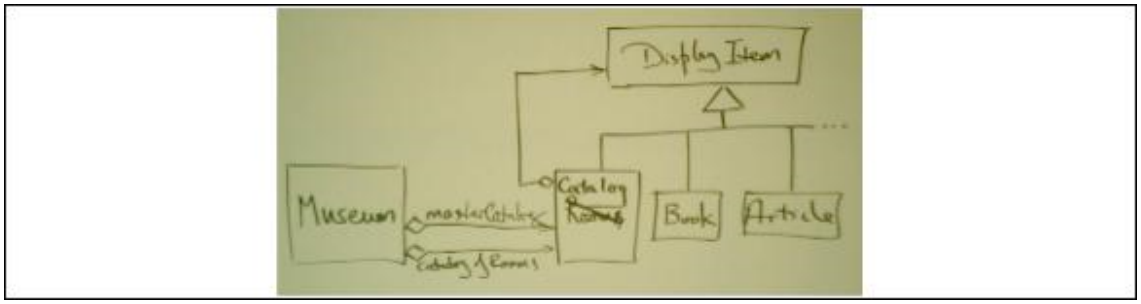


Figura 5-36. No papel. (Graças a Alistair Cockburn)

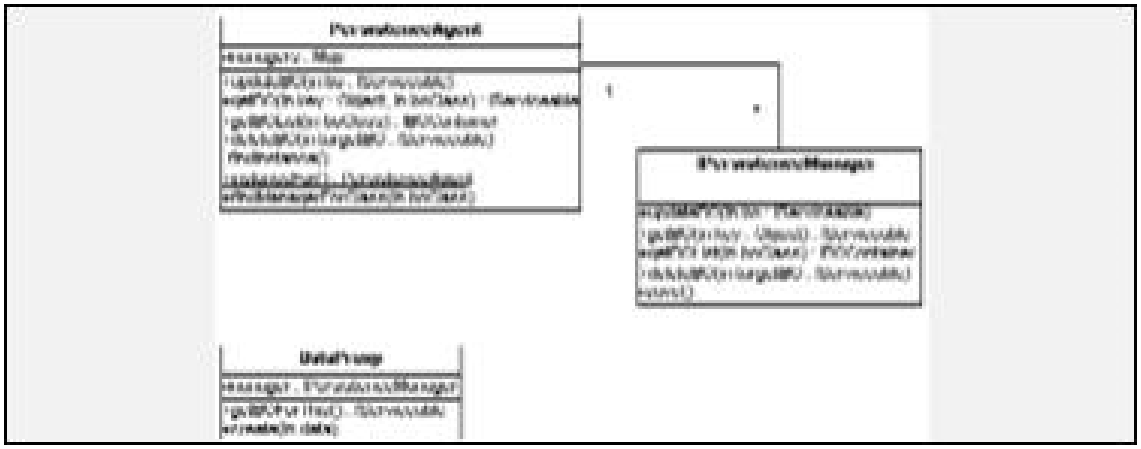


Figura 5-37. Em uma ferramenta CASE. (Graças a Jonathan House)

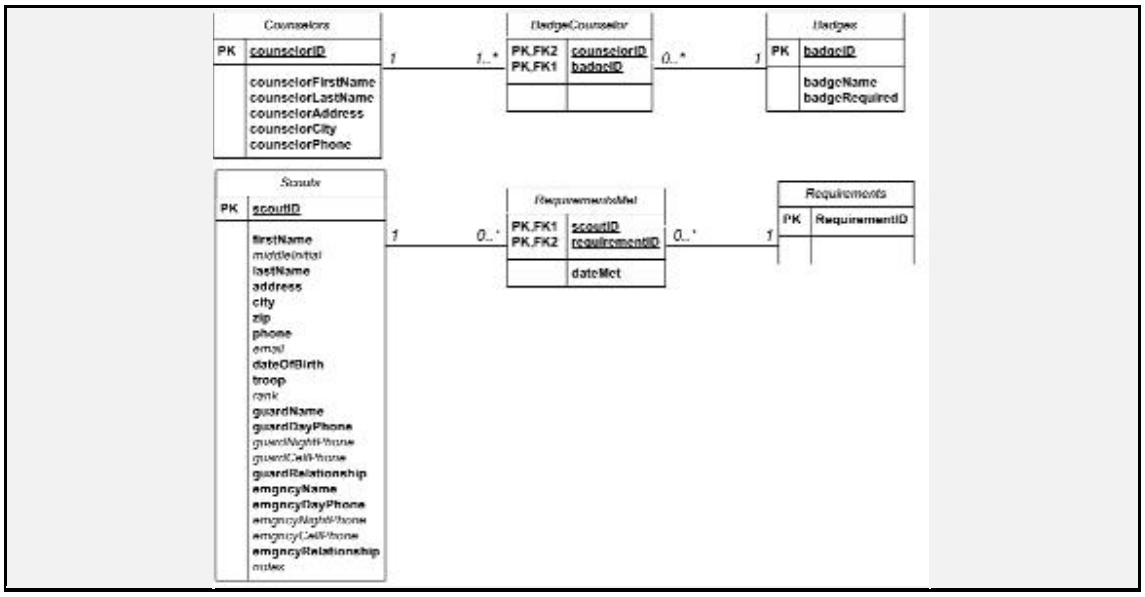


Figura 5-38. Um modelo de domínio não-OO (O modelo de domínio de BSA)

Designer-Programmer: **Source Code e Pacote de entrega**

Eu tenho que incluir essa página apenas para que as pessoas não acho que eu nunca esquecer que os programadores designer-realmente produzir pacotes de código-fonte e de entrega, além de todos os vários itens utilizados para planejar, acompanhar, testar e descrevê-los.

O Crystal Clear não regular o idioma, nomeando, formatação e comentando convenções que você usa. Essas são para a sua equipas para decidir.

Espero que você já sabe o código fonte se parece. Se você realmente quer olhar para alguns, vire à exemplo de teste JUnit :-)

***Designer-Programmer:* design Notes**

Crystal Clear requer "esboços e notas, conforme necessário." Esta é uma frase em aberto, permitindo que as equipas para escrever memorandos técnicos, criar diagramas UML, criar teias wikiwiki descrevendo seu projeto (wikiURL), tirar fotografias de seus quadros, ou fita de vídeo mini-palestras sobre o projeto.

As notas de design são produzidas de forma contínua ao longo do projeto pela *DesignerProgrammers.*

Em termos da teoria de desenvolvimento de software (ver *Questionada),* notas de design servem um dos dois propósitos e têm diferentes paybacks no jogo económico-cooperativo.

- **Eles podem servir para *lembrar,* como parte do *ávido* conjunto de ações, visando a primária, a meta de curto prazo de entregar este sistema em breve.**
- **Eles podem estar entre os *investimento* ações, visando o objetivo secundário de criar para o próximo jogo.**

Mesmo dentro do conjunto de ações ganancioso, *lembrete* marcadores servem um serviço valioso em lembrar às pessoas o que eles decidiram anteriormente. Sendo apenas para as pessoas que estavam presentes (incluindo-se um mês ou dois mais tarde), eles são naturalmente atraídos em guardanapos, cartazes e afins. Estes marcadores raramente precisa ser redesenhado (e pode até perder informações quando redesenhada).

***informando* marcadores são criados como parte da actividade de investimento, para ajudar aqueles que vêm mais tarde para recuperar o atraso para o grupo. Eles são mais longo e mais trabalhoso para produzir. O retorno económico**
é **que a mesma informação não tem de ser repetido uma e outra vez pelos desenvolvedores seniores ocupadas no projeto.**

É **útil para manter várias coisas em mente quando decidir o que e como produzir estes marcadores. Primeiro, ele nunca será possível para transmitir toda a teoria do projeto por meio desses documentos. Em segundo lugar, até mesmo as pessoas que estiveram no projeto do começo não têm uma compreensão compartilhada ou completa do que está no código e que a teoria melhor descreve. Isso significa que as notas de design nunca pode ser completa, consistente e totalmente instrutivo. Portanto, não acho que é o seu objetivo de torná-los tal. Seu objetivo é levá-los *perto o suficiente* para que eles possam fazer boas perguntas, ou desenvolver sua própria teoria o suficiente para que eles possam entrar no código e saber mais sobre o seu próprio (e, esperamos, para que a teoria que venha com não é muito distante de seu próprio).**

Bons esboços e notas deixam um rastro de outro *Designer-Programmer* seguir. Os melhores descrever não como o sistema parece atualmente, mas por que certas escolhas foram feitas e outros rejeitados.

Parte de ambos lembrando e a informação é reter e transmitir a história das escolhas que foram feitas. Recentemente, algumas pessoas têm relatado que o uso de um tipo Yahoo Egroup do sistema de e-mail acorrentado, ou um sistema do tipo newsgroup como Starteam com notas de rosca é útil desta forma. Estes sistemas permitem que as pessoas a seguir a cadeia

de discussão sobre vários temas específicos. Eu adicionar a nota que a cadeia da história é mais informativo, tanto lembrando e informando que é a soma no final da cadeia. Este método é claramente adequado para equipas maiores e distribuídos. Quando a equipas consiste apenas de 3-5 pessoas localizadas todos em um quarto, pode ser menos natural para que as pessoas on-line para adicionar essas notas.

Há um custo contínuo para criar, alterar e manter estes documentos como o próprio projeto evolui, e um custo competindo para não criá-los. Apenas como isso tensão é resolvida é até a equipas, *incluindo o Patrocinador,* que deve considerar o equilíbrio entre o curto prazo e a saúde a longo prazo do sistema. É improvável que o *Patrocinador* vai ser feliz com nenhuma documentação em tudo para seguir equipas, e assim por Crystal Clear exige *alguns* Design Notes a ser produzido.

Aqui estão alguns exemplos de esboços e notas.

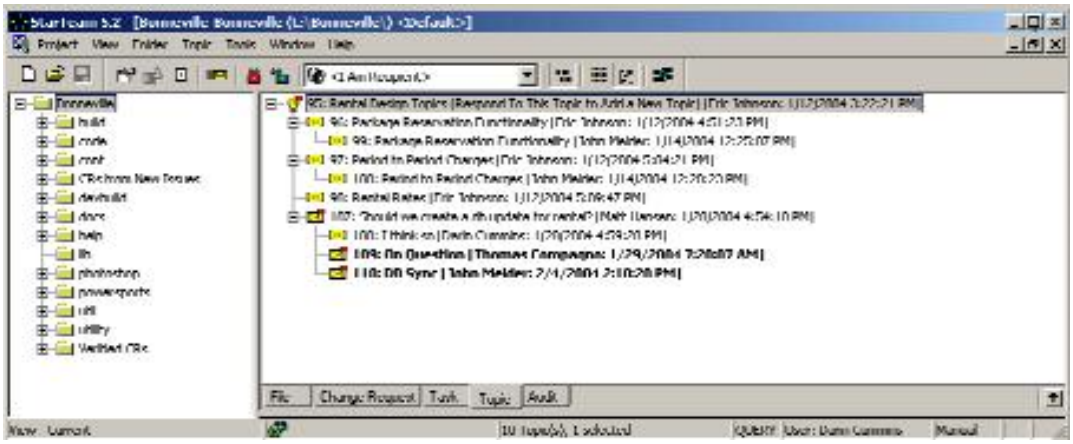


Figura 5-39. Captura de tela do sistema Starteam-style newsgroup, mostrando a trilha discussão encadeada. (Graças a Darin Cummins)

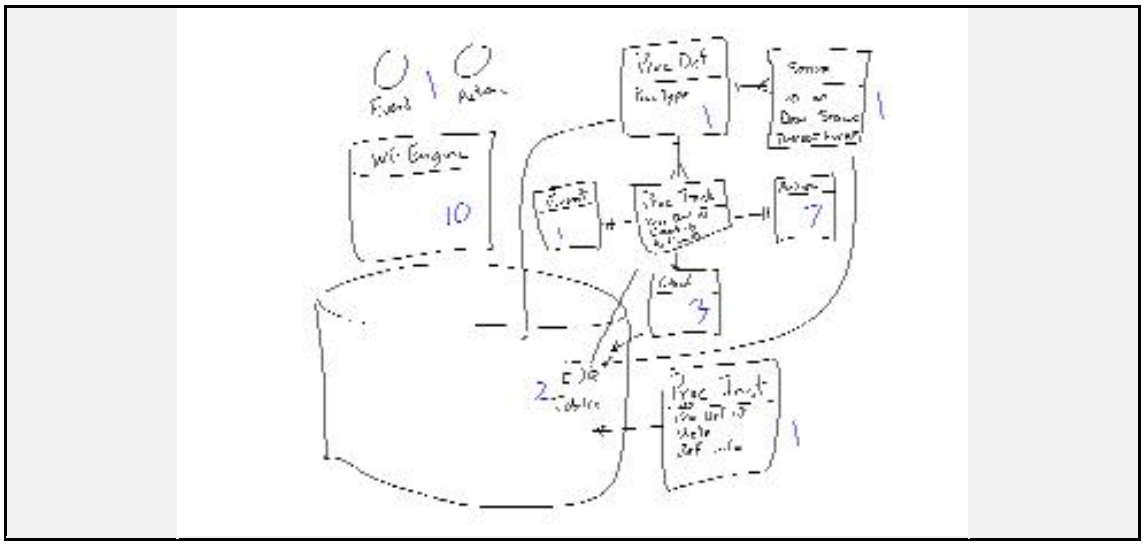


Figura 5-40. UMA *lembrete* nota projeto capturado com o dispositivo Mimeo ligado a um quadro branco e um laptop. (Graças a Darin Cummins, que diz que periodicamente puxa para cima isso e semelhante quadro Mimeo captura para re-exame)

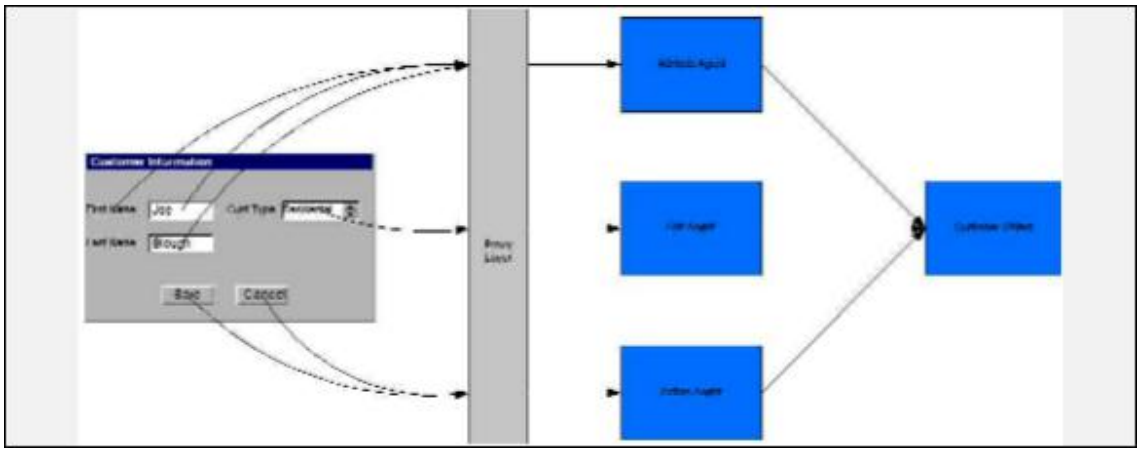


Figura 5-41. A *informando* nota sobre ligação componentes visuais. (Graças a Jonathan House)

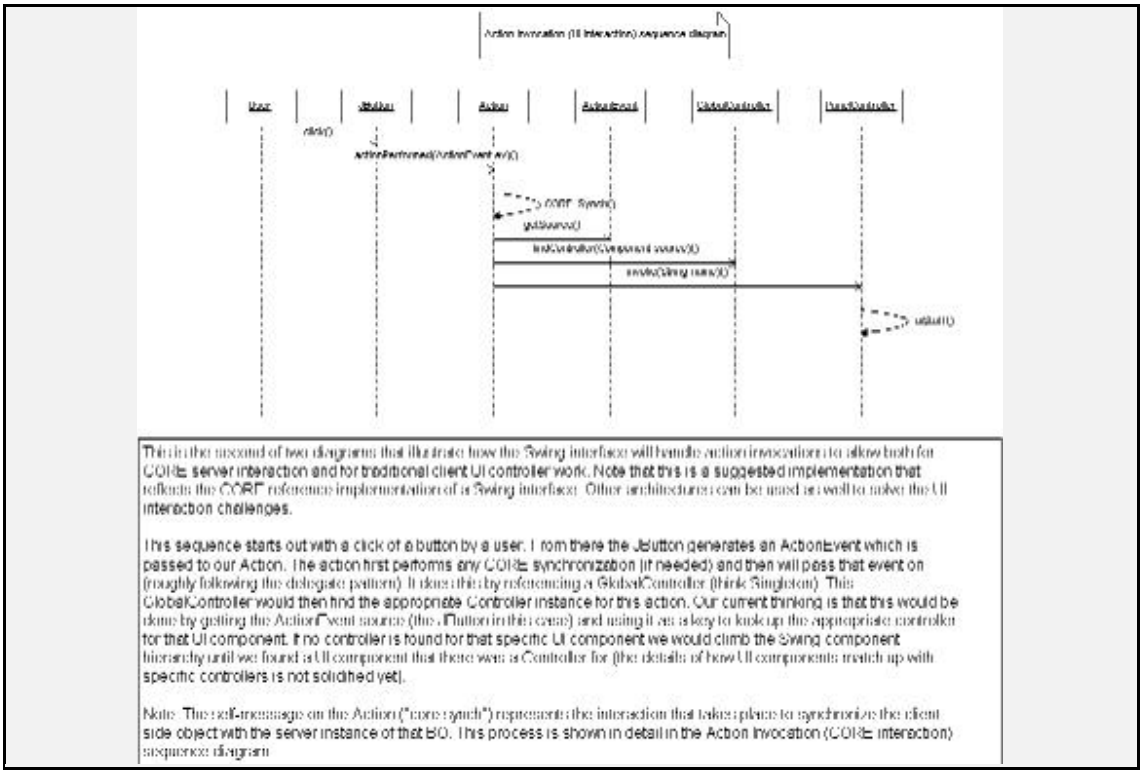


Figura 5-42. diagrama de sequência anotada com uma descrição textual, parte de um domínio série fragmentos modelo misturados com diagrama de sequência e texto explicativo, cada secção iluminar apenas um aspecto da concepção. (Graças a Jonathan House)

***Designer-Programmer:* testes**

Um teste é de preferência executável, *código* que testa um aspecto do sistema. testes executáveis não envolvem uma pessoa que segue um documento escrito que descreve o que a pessoa deve fazer para testar o sistema. Há momentos em que estes são necessários, também.

Os testes são produzidos de forma contínua durante cada iteração. Embora possa haver um grupo de teste externo, gostaria de destacar que não é aceitável para o *designer-programadores* apenas para escrever código e entregá-la a um grupo de teste para reparar. A criação e a execução de testes é uma parte intrínseca do *Designer-Programmer* descrição do trabalho.

Estes dias, com a disponibilidade do equipamento de teste da unidade JUnit e suas variantes, CppUnit, VBUnit, StUnit e assim por diante, há pouco desculpa para não escrever testes de unidade automatizados. testes de aceitação de condução GUI são mais difíceis de automatizar. Bons designers encontrar maneiras de automatizar, pelo menos, testes de manipulação de dados, se não do mouse, botão-e testes de teclas. O equipamento de teste HttpUnit e é parentes ajudar a automatizar testes de interfaces web. Os quadros FIT e FitNesse 45 ajudar as pessoas de negócios, e não apenas programadores, escrever testes de aceitação orientadas a dados. Há um bastante forte literatura sobre o teste automatizado, incluindo livros sobre design orientado a testes (Beck 02 ??, Astels 03, Thomas 04)

Um bom teste de unidade não só testa o que é suposto fazer, mas tem um nome que mostra que "preocupação" está sendo testado (o nome do teste lê como "teste *este* algo é verdadeiro "). Atender a legibilidade em nomear a preocupação significa que a coleção de nomes de casos de teste ilustra um conjunto de preocupações que entraram em projetar o sistema. Essa lista serve como uma parte da documentação de projeto para a próxima programador junto (ver o segundo exemplo, abaixo)

Testes boa aceitação são legíveis com o especialista de domínio, bem como testar os casos normais e de contorno do sistema. Este legibilidade serve para informar o próximo especialista de domínio junto.

Aqui estão dois exemplos de testes JUnit, cortesia de Andy Pols e Jeff Patton. Observe o uso de nomes de teste baseada preocupação em ambos. No segundo exemplo, eu incluo os nomes de cinco casos de teste para que você possa ver como a lista de nomes de servir como *informando* marcadores de preocupações de design. (No primeiro exemplo, a mensagem assertEquals dirá o leitor relatório de ensaio que deu errado; no segundo exemplo, a mensagem assertEquals está dizendo o programador que a preocupação está sendo testado aqui Ambos os estilos de trabalho..)

45 http://fitnesse.org

```
public void testInitialFormDefaultsToUseTheAntBuilder () throws Exception {
    // executar
    Resultado cadeia = myAction.doDefault ();

    // verificar
    assertEquals ( "Deve ser no modo de entrada, uma vez que o formulário foi inicializado",
                                                Action.INPUT, resultado);
    assertEquals ( "Deveria ter formiga como o construtor padrão no formulário de entrada",
                                                  "Ant", myAction.getBuilder ());
}

public void testPreventsAddingDuplicateProjects () {
    // configuração
    Corda duplicateProjectName = BeetleJuiceStub.DUPLICATE_PROJECT_NAME;

    // executar try {

       myAction.setName (duplicateProjectName); falhar (
      "Deveria ter jogado uma exceção"); } Catch
      (IllegalArgumentException IAE) {
        // verificar
        assertEquals ( "Deveria ter retornado a mensagem de erro forma correta",
                                              "Este projeto já existe.", Iae.getMessage ());
    }}
```

Figura 5-43. Os casos de teste com nomeação "à base de preocupação". (Graças a Andy Pols)

```
 classe pública DatabaseModelTest estende FrameworkLocalTestCase {
        public void testDataModelCanBeCreatedFromClientDirectives
        () {
        public void
        testDataModelCanBeCreatedFromDirectivesFoundInXMLFile () { public
        void testDataModelCorrectsExistingSchema () { public void
        testTableCanDropItself ()
        public void testDataModelCanDropItselfFromDB () {

// Apenas o teste ampliado aqui ...
        public void testDataModelCanBeCreatedFromClientDirectives () {
                DatabaseModel dm = new DatabaseModel ( "test"); assertEquals (
                "databaseModel foi criado e nome está correto",
                                                "Teste", dm.getName ());

                Tabela tb = nova tabela ( "primário");
                assertEquals ( "tabela foi criada", "pai", tb.getName ());

                assertEquals ( "modelo de banco de dados não tem mesas",
                                                0, dm.getAllKnownTables () tamanho ()).;
                dm.addTable (TB);
                assertEquals ( "modelo de banco de dados tem uma tabela",
                                                1, dm.getAllKnownTables () tamanho ()).;
                assertEquals ( "table conhece o seu modelo de pai",
                                                dm, tb.getDatabaseModel ());

                Coluna col = new coluna ( "parentId", java.sql.Types.VARCHAR, 20); assertEquals
                ( "tabela não tem colunas", 0, tb.getAllColumns () tamanho ().); tb.addColumn
                (col);

                assertEquals ( "tabela tem uma coluna", 1, tb.getAllColumns () tamanho ().);
                tb.addColumn (col);
                assertEquals ( "mesa ainda tem uma coluna após double add",
                                                . 1, tb.getAllColumns () tamanho ());
        }
```

A Figura 5-44 Os casos de teste com nomeação "à base de preocupação". (Graças a Jeff Patton)

Testador: **Relatório de erro**

Um relatório de bug é uma nota registrando um erro, dizendo que circunstâncias causou o erro. Automatizados equipamentos de teste, tais como JUnit produzir automaticamente tais relatórios. sistemas de construir e testes automatizados, como Cruise Control (cruiseControlURL) automaticamente produzir e enviá-las para o proprietário de teste designado. executar manualmente testes de aceitação exigem que o testador tipo de relatórios de teste durante os testes de aceitação para que a equipas pode voltar e alterar os requisitos, design e código conforme necessário.

Muitos projetos Crystal Clear não tem nenhuma equipas de teste externo dedicado. Os utilizadores se tornam os testadores do sistema. Mesmo nesses casos, eles relatam bugs. Um sistema de relatório de bug open source popular é Bugzilla e há outros de código aberto a ser desenvolvidas (por exemplo, http://projects.edgewall.com/trac fornece Trac, que usa a tecnologia WikiWiki para relatórios de erros e rastreamento).

Um bom relatório de erros fornece informações suficientes para fazer o teste fácil de recriar. Aqui está um exemplo de um relatório de bug digitado manualmente a partir do projeto BSA.

Test Incident Report	*Scout 01*
Test Case Identifier	*AddNewScout*
Incident Description	
Date:	*14-Apr-03*
Tester(s):	*All*
Actual System State:	*Code has a counter for the scoutID, but the database had an autocounter and could not be manually inputed.*
Actual Output:	*When clicking of New Scout, an error message would pop up stating that scoutID field could not be changed.*
Defects Detected:	*The scoutID had an error and system crashed when trying to save a new scout.*
Test Environment:	*Regular Regression Testing*
Attempts to Repeat:	*Ran the test twice with same error and crash happening*
Anticipated Impact	*No scouts can be added into system, therefore making the system useless.*
Resolution	*Problem was found, and the autocounter in the database was taken out so the counter in the code could work correctly.*

A Figura 5-45 (Graças ao projeto BSA)

	Módulo	bug Descrição	estado	orig
	Cliente	Início ligação produz um erro 404 a partir da página widget.html. Ligado ao home.htm e deve ser ligada a index.html.	Abrir	NJ
	Cliente	A caixa de entrada taxa de atualização aceita valores não numéricos. fresh every 0.r4a min. OK	Abrir	CA
	camada de negócios	Receber o seguinte erro ao salvar uma ordem de compra. busObj.PO dividir por 0.	Abrir	NJ
	Serviço de internet	.arquivo de disco não foi encontrado.	Abrir	NJ
	camada de dados	Não é possível executar procedimento app_getapo. lista de parâmetro inválido.	Fechadas	CA
	Base de dados	Ao procurar por fornecedores os tempos de resposta após 30 segundos.	Fechadas	CA

Figura 5-46. Criar relatório. (Graças a Nate Jones)

***Escritor:* Ajuda Texto, Manual do utilizador, e manual de treinamento**

Presumo que você já viu texto de ajuda, manuais de utilizador e manuais de treinamento antes em sua vida, por isso não vou insistir no ponto, descrevendo o que se é. Eu incluí-las na lista de produtos de trabalho para ser completo. A equipas terá que decidir não apenas quem escreve cada um, mas também o quanto ele deve ficar escrito para cada iteração e entrega.

manuais de utilização de papel estão cada vez mais sendo substituído por linha no texto de ajuda, por isso pode ser que, do ponto de vista de trabalho-produtos, os dois são equivalentes. Remediar a inutilidade geral da mais-line com a ajuda de texto ("Para activar a hifenização, clique na caixa 'Ligar hifenização') está fora do escopo de Crystal Clear.

I atribuir esses produtos de trabalho para o *Escritor* papel, porque algumas equipas podem contratar ou contratar uma pessoa externa para escrevê-los. Outros podem atribuí-los aos membros da equipas.

Por razões óbvias de tamanho, eu não incluir um exemplo aqui.

Reflexão sobre os Produtos de Trabalho

Que é um conjunto de produtos de trabalho que eu sinto que posso defender como um grupo, é bastante leve e ainda seguro, e que você deve ser capaz de se adaptar à sua situação. Alguns desenvolvedores de quebrar em um suor quando vêem uma lista de todos os produtos de trabalho que são produzidos em um projeto, então eu espero que não doeu muito.

Quero agradecer as pessoas que contribuíram as amostras de produtos de trabalho. Eles forneceram uma ampla gama de formatos diferentes para sua consideração. Você viu que algumas pessoas usam padrão, a documentação formal, alguns quadros de uso, cartazes, cartões de índice e notas pegajosas, e algumas anotações uso personalizados criados com vários tipos de ferramentas. Cada grupo encontrou uma maneira de transmitir a informação necessária com clareza e economia.

Vale a pena revisitar duas ideias.

- **Não há um único, necessário, núcleo ou conjunto mínimo de produtos de trabalho em Crystal Clear, o caminho até lá é com as sete propriedades. Tomando a intersecção de todos os projetos de sucesso que visitei rendimentos muito pequeno um conjunto geralmente seguros, tendo os rendimentos de união muito grande um conjunto (inseguro devido à sua massa). O conjunto neste capítulo deve ser perto do que você precisa.**

- **O jogo económico-cooperativo envolve *dois* objetivos: entregar este sistema, e se preparando para o próximo jogo. Crystal Clear é para pequenas equipas comunicar de perto, com *ambos* objetivos em mente.**

Atender ao primeiro objetivo significa que os produtos de trabalho intermediários deve ser rápido e barato de produzir, com uma ênfase principal no *lembrando* marcadores. Eles devem atender a *informando* marcadores, se novas pessoas serão adicionadas à equipas dentro do projeto, porque eles podem sair na frente, mesmo dentro de um horizonte de tempo de seis meses por gastar tempo criando um caminho de aprendizado para novos companheiros de equipas a seguir, para que eles não pedem bastante tantos perguntas para as primeiras semanas.

O segundo objetivo é a criação para o próximo jogo. Há novamente uma dupla ênfase: melhorar as habilidades dos membros da equipas para que eles estarão melhor na próxima rodada, e estabelecendo no lugar *informando* marcadores para a próxima onda de desenvolvedores. Não se esqueça que as pessoas da equipas são eles próprios altamente eficaz "informando marcadores" e "radiadores de informação." Construir uma estratégia para a formação da próxima onda de desenvolvedores que usam as pessoas existentes (com as novas pessoas que emparelhar programação em rotações com os membros da equipas existentes constitui uma tal estratégia).

Finalmente, há várias partes interessadas na questão da documentação. Os desenvolvedores atuais estão interessados em um conjunto claro de marcadores, lembrando principalmente queridos. Os futuros desenvolvedores estão interessados em marcadores informativos (e não tão chato) para recuperar o atraso para o grupo. E a *patrocinadores* estão interessados na estabilidade a longo prazo da base de conhecimento.

Nenhuma dessas partes interessadas é para ser esquecido. Eu acredito que com um pouco de criatividade, você pode chegar a formas ainda mais expressivos, menos caros e menos dolorosas para satisfazer seus interesses combinados.

Capítulo 6

Incompreendido (Erros comuns)

Você acha que você está usando Crystal Clear, mas seu projeto não está funcionando. O que há de errado? Crystal Clear pode falhar, mas vamos primeiro verifique que você realmente está fazendo Crystal Clear. Este capítulo apresentam situações de projeto de exemplo. Alguns deles cumprir a intenção de Crystal Clear, outros violá-la. O objetivo aqui é para lhe fornecer um sistema de alerta pessoal que você é ou não estão em sintonia com a intenção de Clear.

Erros de interpretação do Crystal Clear é inevitável, não importa quantas palavras que eu escrevo. Uma maneira de ajudar a reduzir os erros de interpretação é discutir situações e perguntas específicas que surgiram.

Aqui estão algumas situações que tenho pela frente. A primeira série indicam clara violação da Crystal Clear, a seguinte série discutir violações de intenção e situações limítrofes.

"Nós colocated e correu iterações de duas semanas - por que nós falhamos"

A comunidade de desenvolvimento de software atualmente excessivas a palavra *iteração 46* e apreciação insuficiente o *do utilizador.* A literatura acentua *iterações* ao invés de *as entregas aos utilizadores reais.* XP exige um *no local do cliente,* que, embora louvável, é muitas vezes tão difícil de arranjar que as organizações simplesmente jogar acima de suas mãos e não envolvem um utilizador real.

Lembre-se que a propriedade # 1 de Crystal Clear é Entrega frequente, não *iteração frequente,* e Propriedade # 6 é Fácil acesso a utilizadores experientes.

O assunto foi levado para casa para mim no outro dia quando eu visitei um grupo cujos hábitos derivado do XP. Eles sabiam sobre a programação em pares, design orientado a testes, testes de regressão automatizados, iterações curtas, e todo o resto. Eles tinham evoluído ao longo do ano, como é muitas vezes o caso, para algo semelhante a Crystal Clear. Nós estávamos fazendo uma *workshop de reflexão* no momento desta história. Metade das pessoas na sala foram liquidação de um projeto de cinco meses. Eles mantiveram referindo-se a uma parte do sistema que era "simplesmente errado", apesar do fato de que ele trabalhava e tinha um bom conjunto de testes de apoio.

> **Traçamos um cronograma de suas práticas em relação ao ano anterior. Eles se reuniram suas exigências de que era supostamente um conjunto bem informado sobre as partes interessadas. Depois de três meses, eles tinham implantado uma versão inicial do sistema para o site do cliente utilizando o Web-X, e tinha mantido uma discussão online do sistema. Tudo correu bem neste implantação prática.**
>
> **Após o quarto mês, uma semana antes de nossa pequena oficina, cinco futuros utilizadores do sistema veio para o desenvolvimento local e sentou-se com os desenvolvedores para a prática de utilizar o sistema. É quando eles disse aos desenvolvedores que era "errado". Você pode imaginar isso chocou a equipas de desenvolvimento não tem fim. A implantação Web-X de alguma forma não tinha feito o mesmo exame detalhado do software como a visita.**

---

46 Nos últimos discussões com grupos de utilizadores ao redor do mundo, nós estamos encontrando a frase "desenvolvimento iterativo" em si é problemática, porque "iterativo" implica retrabalho, o que muitos patrocinadores encontrar ameaçando desde o início. A alternativa, "desenvolvimento incremental" é muito mais tranqüilizador, pois implica "acrescentando em." Para não constantemente lutando termos padrão, adotei a palavra padrão da indústria *iteração* neste livro para se referir a um período de desenvolvimento interno. I ainda se referem à estratégia global como *desenvolvimento incremental* com *entrega incremental,* para reduzir o stress na mente dos patrocinadores.

> **Nós discutimos o que poderia ter feito de forma diferente e que eles poderiam fazer diferente no futuro. Para nossa grande surpresa mútua, a equipas descobriu que eles nunca realmente tinha tido acesso a um "user friendly" que poderia levar entregas provisórias e examinar o sistema de cultivo, e pior, eles ainda não podia ver que eles poderiam obter um no próximo futuro.**
>
> **Em outras palavras, olhando para trás em sua linha do tempo, eles não poderiam ver como eles poderiam ter evitado este grande surpresa.**
>
> **A história tem um final feliz, eu estou feliz em dizer. Este foi um grupo produtivo, eles ainda tinha um mês para a esquerda, para que eles eviscerado a peça "errado" e reescreveu-lo para atender às necessidades dos utilizadores.**

Esta equipa tinha escutado seus patrocinadores e utilizadores, colocated, integrada com frequência, e refletida após cada uma das suas iterações, e eles não tinha visto isso chegando. Eles ainda tinham nenhuma maneira de corrigi-lo. Essa responsabilidade recai sobre o *Patrocinador executivo,* quem é a pessoa que tem de organizar para mais visitas de utilizadores reais. Felizmente, neste particular *Patrocinador executivo* está ciente da importância da questão e está activamente empenhada em melhorá-lo para projetos futuros.

Iterações fornecer feedback para a equipas sobre seu processo, ganha cedo para a sua moral, e um mecanismo de direção para ver que eles permanecer na pista. Mas eles não fornecer feedback end-to-end sobre a aptidão do software para uso comercial. Isso requer o envolvimento de utilizadores reais.

Se eu tivesse que escolher entre iterações de duas semanas sem acesso para os utilizadores reais, ou dando-se as iterações curtas e entregando apenas trimestral, mas com utilizadores reais entrar no laboratório de desenvolvimento duas vezes no período de entrega de rever software crescendo, eu escolheria o último . Esta é a razão para o produto de trabalho *Visualizando Programação,* e a razão que insisto *Entrega* ciclo longo *iteração* ciclo.

"Dois desenvolvedores estão separados por um corredor e uma porta trancada."

"Oh, vamos lá, Alistair, o quanto isso realmente importa? Não podemos chegar em nossas pessoas no mesmo espaço corredor. Acontece que temos um fechamento de combinação em que porta do corredor."

Não importa, se você quiser a segurança oferecida pelo Crystal Clear. Crystal Clear baseia-se em pessoas trocando pequenos pedaços de informação em uma alta frequência ao longo de um dia, sem perder a sua capacidade para completar a tarefa em mãos. Essa velocidade de comunicação cria uma rede de segurança para o projeto.

Suponha que uma pessoa pode levantar-se, olhar sobre uma partição de baixo, perguntar e, em poucos segundos obter a resposta a:? "Será que você realmente quer dizer que os utilizadores nunca terá que digitar novamente a senha Temos três servidores em uso, cada um com a sua sistema de senha própria." Seu tempo total gasto longe da tarefa em mãos é talvez 45 segundos. Sua capacidade de continuar com a tarefa em mãos é excelente.

Contraste isso com a mesma pessoa que pensa: "Eu não sei se eu quero parar com isso apenas ainda para atravessar o salão, soco na combinação. Talvez Pat não está em sua mesa agora. Vou vê-la em algum momento hoje." O resultado é ou nenhuma resposta em tudo, ou vários minutos, envolvendo um número de diferentes sub-tarefas e uma grande chance de se distrair, passou longe da tarefa em mãos, pelo menos. Sua capacidade de continuar com a tarefa em mãos é moderado na melhor das hipóteses.

Ao negociar o corredor, escadas, fechaduras, uma pessoa tem de lidar com sub-tarefas que se intrometem em seu estado cognitivo da mente. Isso quebra a linha de pensamento por um período e faz voltar para o consumo de energia em estado de resolução de problemas. Lidar com essas mudanças cognitivas retarda o fluxo de informações entre indivíduos várias ordens de magnitude, bloqueando as trocas que fazem uma equipas produtiva e bem sucedida.

É **verdade que muitos projetos enviados software apesar desenvolvedores sentado do outro lado corredores. No entanto, os líderes de equipas repetidamente me dizem: "Dê-me 3-5 desenvolvedores em um quarto, manter as distrações de distância, e eu posso obter o software para fora. Não espalhe-nos para fora." Isso significa que eles visitam uns aos outros, desenhar em cada outros quadros brancos, olhar por cima uns dos outros ombros em seus programas e testes.**

"Temos este grande infra-estrutura para entregar em primeiro lugar."

Às vezes é preciso um longo tempo para obter a infra-estrutura criada para entregar apenas o primeiro menor pedaço de funcionalidade,. É por isso que o período de incremento permitida é tão grande quanto 3 meses. Mas maiores do que isso, e os chefes de projeto dizem-me que não pode manter o foco no projeto em mãos, que a possibilidade de erro no projeto é muito grande. Cada vez mais, as equipas estão querendo entregar novas funcionalidades em 3-6 semana prazos, dizendo que eles não querem esperar mais tempo do que isso para obter feedback. Eles implementar uma versão magro da infra-estrutura primeiro, e depois evoluir a infra-estrutura em paralelo com a funcionalidade (o *caminhando de esqueleto* e *rearquitetura incremental* estratégias).

Crystal Clear é baseado em obter feedback de execução de código e de utilizadores ativos. O feedback não é apenas sobre o código, mas sobre os requisitos e processo de desenvolvimento, sobre a capacidade de sua equipas para entregar sistemas.

feedback rápido sobre o código permite que a equipas de reduzir ainda mais escrevendo requisitos e desenhos em detalhes excruciantes. Se você prolongar o tempo antes de você ter executado, testado, examinado código, e você tem que colocar mais em escrever apenas para manter a memória viva. feedback rápido sobre o processo permite que a equipas para localizar problemas inesperados e configurar uma ação corretiva. Ele também aumenta a moral e estabelece o hábito de entregar que é um dos fatores críticos de sucesso de equipas de trabalho.

iterações curtas e ciclos de entrega são benéficos por quatro razões:

- **para evitar ter de escrever tantos detalhes de design,**
- **para obter feedback real sobre as decisões de design,**
- **para encontrar e corrigir problemas no processo de desenvolvimento, e**

- **para estabelecer um hábito de entregar.**

"Nossa primeira entrega é uma demonstração das tabelas de dados."

Nada de bom. Cada ciclo de entrega deve produzir execução, testado código que é, pelo menos em princípio, de alguma utilidade para alguns utilizador.

Movendo-se para o desenvolvimento ágil requer aprender a dividir a atribuição desenvolvimento em pedaços que podem ser integrados de ponta a ponta, exercendo o processo de desenvolvimento integral e, como grande parte da arquitetura como é necessário para a funcionalidade crescendo.

É **uma armadilha reconfortante para desenvolver o design da interface do utilizador, ou nas mesas, ou um pequeno protótipo, e chamar isso de "um ciclo" (de algum tipo) após o qual um *workshop de reflexão* é mantido. Tenho ouvido falar de equipas que chamado escrevendo todos os requisitos "uma iteração," e, em seguida, refletiu sobre seu processo de obtenção de requisitos.**

Isso não funciona para as duas razões que eles não podem saber o quão bem eles se reuniram e documentados os requisitos até que vejam como tudo engrenado com o resto do seu processo de desenvolvimento (isto é, após o parto).

"Nenhum utilizador está disponível, mas temos um Engenheiro de Teste se juntar a nós na próxima semana."

Um engenheiro de teste não é um utilizador. Nem é um gerente ou supervisor que "uma vez trabalhou no campo." Seu utilizador experiente valida e quebra seus projetos de interface do utilizador. Um engenheiro de teste pode verifique seus próprios testes para dizer-lhe se o seu código é quebrado, mas não posso dizer se o sistema é o que os utilizadores querem. Em um estudo publicado de gerentes de projeto, cada um dos quais tinha experiência com e sem links diretos para os utilizadores, os gerentes de projeto esmagadoramente sentiram que os projetos com ligação direta do utilizador foram mais bem sucedidos do que aqueles sem (Keil 95). Quando eles tinham utilizadores presentes, os utilizadores alertaram eles, logo no início, para erros que eles estavam fazendo no conceito e layout do software, permitiu-lhes produzir software mais adequado para as necessidades dos utilizadores, com menos recursos desnecessários.

Um projeto Crystal Clear pode conviver com produtos de trabalho intermediários relativamente informais, devido à capacidade dos projetos de simplesmente pedir a um utilizador, em intervalos não superiores a uma semana de intervalo. Os melhores projetos têm um utilizador disponível em tempo integral, e alguns podem obter o utilizador a visitar várias manhãs por semana. No mínimo, uma hora a cada semana deve ser agendada para a entrevista utilizador ou tempo de projeto compartilhado.

Essas foram as violações óbvias. Aqui estão um conjunto de questões e situações que, ou violam a intenção de Crystal Clear ou questionam o limite.

"Um desenvolvedor se recusa discutir o seu projeto ou mostrar o seu código para a

descansar."

Como disse desenvolvedor especialista Pete McBreen, "Os dias de alegando que código é privado estão muito longe." Disponibilidade para discutir o projeto e encontrar possíveis falhas em sua / seu próprio design ou código é importante.

Pessoas sendo dispostos a discutir o seu código é parte do Segurança pessoal propriedade. Os programadores muitas vezes discordam sobre o que constitui um bom design, e muitas vezes são bastante desagradável sobre ele. Proporcionar às pessoas com a necessária Segurança pessoal pode exigir que algumas pessoas passo em frente e mostrar seus projetos para que a equipas pode aprender a discutir o projeto tecnicamente e não emocionalmente.

Um projeto é improvável que retirar um processo de Crystal Clear se eles têm medo de confiar em seus projetos para seus companheiros de equipas. Pergunte a si mesmo como você e eles podem gerenciar uma discussão de design sem insulto, deixando as pessoas diferentes trabalhar em seus próprios níveis de competência, ainda suficiente para entregar um sistema de trabalho.

"Os utilizadores querem todas as funções entregues a suas mesas ao mesmo tempo..."

"... nós não quero irritá-los por ter que instalar novas versões tantas vezes." Isso acontece com certos tipos de sistemas que os utilizadores não pode ser perturbado com muitas entregas de sistema de mudança e, especialmente, mudando interface do utilizador. No entanto, você deve ser capaz de trazer um no utilizador para ver e usar o sistema para obter feedback sobre suas decisões de design técnicas e de uso. Uma vez que parte do sistema foi projetado, testado, vistos e aceitos, você pode relaxar sobre essas decisões e passar para a próxima.

Na situação que não é apropriado para realmente instalar o software em máquinas dos utilizadores, definir o construída, funcionalidade testada para o lado (devidamente versão, é claro), e esperar para adicionar funcionalidade do próximo incremento. Em outras palavras, você irá desenvolver exatamente como se você tivesse entregue as funções para os utilizadores, embora, neste caso, você não fez. A chave para o processo de Crystal Clear é que você começa encerramento e feedback sobre o seu processo de desenvolvimento e implantação, bem como o seu software.

Crystal Clear pode ser usada quando você não pode manter ofertas de atualizações do sistema para todos os utilizadores.
Modificar o processo por encontrar um "user friendly", e implantar as diferentes versões para apenas aquela pessoa.

"Temos alguns marcos menos de um caso de uso e alguns maior."

"... Nossos marcos são como estes: implementar novos comandos de comunicação, implementar tags HTML, implementar classe Blob Alguns deles são dentro de um caso de uso, alguns deles abrangem casos de uso farão os..?"

Implementar classe Blob não é um marco "interessante" no Crystal Clear vocabulário. Conseguir um comando de comunicação implementado e testado é. A implementação de um par de tags HTML implementados, testados e integrados é.

Não há violação óbvia aqui. Continuo a perguntar, "O que conta como uma peça útil de funcionalidade."

"Nós escreveu um conceito básico e design do sistema. Todos nós sentar-se juntos, de modo que deve ser bom o suficiente."

Esta é uma referência a "Você tem uma declaração de missão do projeto, você está usando usage-requisitos baseados, provavelmente, os casos de uso, e você tem uma descrição do projeto do sistema usando uma das muitas técnicas de descrição você pode inventar", questionando quantas requisitos são necessários, e como é necessária muita documentação design.

I encontrar casos de uso úteis no início de projetos (menos críticos como o sistema preenche), e então é claro que eu recomendo-los. No entanto, um número de times como listas de recursos ou formatos de requisitos que não são nem casos de uso, nem listas de recursos. Alguns vivem quase inteiramente por acordo verbal. Se todos se sentem juntos, então ele pode ser o suficiente para que você tenha alguma combinação de uma lista ator-objetivo, e / ou histórias de utilizadores ou caso de uso cuecas ou listas de recursos como suas necessidades "tabela de conteúdos", e isso é suficiente para manter suas conversas no lugar. Não há violação do Crystal Clear aqui, desde que o

Patrocinador e *Embaixador Utilizador* fazem parte do acordo para trabalhar desta forma.

A mesma regra se aplica para documentação de projeto.

Cuidado, porém, para uma pessoa se tornar excessivamente carregado como "expert" sobre os requisitos ou design de domínio ou mestre, e retardando a equipas.

"Quem possui o código?"

Eu costumava ter uma regra no Crystal Clear exigindo que "cada produto de trabalho função, classe e tem um proprietário clara." Durante as revisões deste livro, essa regra caiu baixo o suficiente em prioridade não fazer o corte final. No entanto, a propriedade código ainda é um tema quente e até mesmo perigoso em muitos projetos.

Em muitos projetos, todos podem adicionar código para qualquer classe, com a consequência de que ninguém se sente confortável apagar código de outra pessoa da classe cada vez mais confuso. O resultado é algo muito parecido com uma geladeira compartilhada por vários companheiros de quarto: cheio de coisas cada vez mais fedorentos que *quase* todo mundo sabe que deve ser jogado fora, mas ninguém realmente joga fora.

Eu vi um grupo de três pessoas sentadas ao redor da mesma estação de trabalho por dias, trabalhando seu caminho através de todas as classes, de negociação que linhas de código para excluir ou mover. Esta situação vem de ausência de um modelo de propriedade, e é caro.

Você tem um modelo de propriedade claro para seus produtos de trabalho se você sabe quem é que pode alterar, actualizar e, mais importante ainda, excluir qualquer parte dela. A resposta pode ser uma pessoa única, qualquer um em uma determinada sub-time, ou mesmo alguém na equipas. Você nunca deve ter para ter a equipas inteira se sentar ao redor da tela e perguntar:

"Quem colocou isso aqui? O que ele está fazendo lá? Será que podemos excluí-lo?" Se você tiver que fazer isso, você sabe que tem uma ruptura no modelo de propriedade.

Uma pessoa me escreveu: "Bom ponto Nós apenas nos encontramos todos sentados ao redor da tela, na semana passada, revendo todas as classes que temos até agora Parece que tinha sido acrescentando, e ninguém remover Levou um dia inteiro apenas para... classificar. Agora estamos organizados com a posse de caso de uso e propriedade de classe. Qualquer pessoa pode fazer uma mudança a curto prazo a uma classe quando bateu modo de pânico, mas essa pessoa notifica o proprietário classe, que pode dobrar verificá-lo."

As pessoas muitas vezes pensam que o XP não tem modelo de propriedade. Isso é falso. XP tem um modelo de propriedade forte: *Qualquer dois pessoas sentados juntos e concordar com ela pode mudar qualquer linha de código no sistema.* O modelo de propriedade é explicitamente comunal, com a verificação de segurança que duas pessoas têm de concordar com a mudança (e também que todos os testes tem que correr!).

A maioria dos projetos Cristalino I visitaram adotam a política "alterá-lo, mas deixe-me saber."

"Podemos deixar que o nosso Engenheiro de Teste escrever nossos testes? Como podemos regressão testar a GUI?"

testes de unidade de regressão são para o *desenvolvedores* para escrever e usar, não o engenheiro de teste. Eles estão lá para que os desenvolvedores podem ir para casa à noite, sabendo que as mudanças que põem em durante o dia não inesperadamente destruir o sistema, fazer mudanças com maior rapidez, encontrar erros mais rapidamente, e dormir melhor à noite.

Programadores tradicionalmente não gostam de escrever testes, e têm uma tendência a passar o código para outra pessoa chamada Tester para testar assim que o código "tipo de corridas." Enquanto isso não é uma violação estrita de Crystal Clear, é uma violação da sua intenção e uma má idéia, socialmente e profissionalmente. Escrever seus próprios testes.

Isso levanta a questão de testar novamente. Avaliação Propriedade 7, que discute GUI- testes menos do sistema, os testes do sistema baseados na GUI, e testes de utilização.

"Qual é a duração da iteração ideal?"

Eu não sei que há um "ótimo" duração da iteração em geral. Ouvi defesa adequada de tudo, desde uma semana a um mês. É útil ter em mente os diferentes efeitos negativos vêm de eles serem muito longos, muito curto, e tudo a mesma coisa.

Demasiado longo. **A maioria das organizações de desenvolvimento usam comprimentos de iteração de qualquer lugar a partir de três meses a dois anos. Isso produz o ritmo tão bem descrito no Scrum Egroup por Daniel Gackle (que estava escrevendo sobre mesmo uma iteração mensal!):**

> **"Meandro, meandro, meandro, perceber que não resta muito tempo, pirar, ficar intenso, OK estamos a fazer. Repita mensal."**

Este "meandro, meandro, pirar, ficar intenso" síndrome é aquele que o desenvolvimento incremental e iterativo é suposto para remover. A pergunta restante é, quanto tempo é muito longo para você?

Duas boas escolhas iniciais são duas semanas e um mês. iterações de duas semanas pode ser bom, porque esta duração do ciclo irá ilustrar aos programadores o quanto eles precisam mudar. Ele vai ensiná-los a estimar em pequenas quantidades e desenvolver-se em incrementos de micro. Pode, no entanto, ser muito difícil inicialmente. A duração do ciclo de um mês é menos de um choque para o sistema e pode permitir que a equipas a facilidade para a nova abordagem. Você pode encontrar, no entanto, que iterações de um mês não são eficientes o suficiente para seus propósitos.

Você vai, naturalmente, ser segurando *Workshops reflexão* após cada iteração, para que possa questionar a duração da iteração dentro de algumas iterações. Alguns grupos de mudar de duas semanas a um mês, outros de um mês para duas semanas. Eu mesmo conheci equipas que mudam a duração da iteração, dependendo de aonde eles estão dentro do projeto! (Para os interessados 47: eles usam iterações de uma semana no início do projeto para evitar a síndrome do "serpentear" descrito acima, mover para iterações de duas semanas à medida que acertar o passo, e voltar para iterações de uma semana como eles perto do fim, a fim para afinar o alcance de corte e lidar com as mudanças de requisitos mais atuais e erros de integração.)

Curto demais. **Iteração start-up e shut-down não é livre, por isso, se as iterações são muito curtos, a equipas vai ficar frustrado em reiniciar tantas vezes. Mesmo as pessoas que gostam de iterações curtas, por vezes, comentar sobre o incômodo de ter que fazer o planificação e estimativa de cada manhã segunda-feira. Além disso, se você não tem testes de integração e aceitação automatizados boas, a carga de testes será muito alto.**

Também em uma discussão Egroup, Patrick Parato observou sobre iterações de uma semana:

> **A única desvantagem que temos visto é que a equipas de desenvolvimento fizzles para fora perto do final da iteração. As tarefas serão todos completos, mas as pessoas não querem admitir isso porque o iterações semana que eles se sintam constantemente sob pressão. Os desenvolvedores realmente não gosto da sensação de que eles não podem variar seu esforço dependendo do humor, saúde ou outros estímulos externos. Alguns desenvolvedores acreditam que um mês inteiro iteração permite-lhes mais liberdade para variar seu ritmo, e que a uma semana iteração é muita pressão. Muito semelhante ao micro gestão.**

---

47 Graças a Jeff Patton para esta nota.

Monotonia. **Nota do Patrick introduz o tema da monotonia. Como mencionado anteriormente, os seres humanos parecem construídas para ritmo, e a necessidade de esta mostra-se em desenvolvimento incremental / iterativo. Um número crescente de pessoas estão relatando que quando o fazem iterações curtas por um ano ou dois, uma sensação de esgotamento em conjuntos, apenas a partir do nivelamento da sua vida de trabalho. Isso ocorre mesmo quando e mesmo que eles estão entregando bom código para os seus clientes regularmente.**

Você pode dar dois passos para prevenir esta monotonia. O primeiro é para atender às suas iteração e de conclusão de entrega rituais (ver a *Processo* capítulo). No final de sua iteração, descomprimir ou comemorar, mesmo que isso signifique um passeio na floresta ou uma viagem de grupo para o pub. Organizar para fazer algo diferente depois de entregar o novo código. Algumas equipas usam o período logo após a entrega para limpar a bagunça que eles criaram no código apenas antes do parto (Hohmann ref ???), outros alocar tempo para algum desenvolvimento pessoal. As pessoas usam a *workshop de reflexão* e planificação de iteração para criar uma mudança de ritmo.

A segunda etapa se aplica se você estiver usando comprimentos de iteração mais curtos, como uma semana. Bundle-los em algum tipo de super-ciclo, talvez a cada quatro ou seis iterações para um ciclo de super. Organizar uma mudança de ritmo para ocorrer nos limites do super-ciclo, um passeio da equipas, piquenique, tratar, ida ao cinema, meio-dia; ou alguns dias para desenvolvimento de temas pessoais.

Algumas equipas de reservar uma quota de tempo para as pessoas a trabalhar em mini-projetos não directamente relacionados com o projeto. Algumas marcas de colocar no código ou no quadro branco sobre lugares feios no código que gostaria de ter tempo para limpar. Alguns alocar tempo para as pessoas a investigar novas tecnologias. Ambos são atividades "investimento", que melhoram os programadores e do sistema, e também proporcionar uma mudança de ritmo.

A idéia final é permitir que programadores de passar tempo com os utilizadores reais de seu sistema em seu ambiente de trabalho. Isto não só quebra a monotonia, desenvolvedores, como Tom Poppendieck e outros apontaram para mim, achar que é altamente motivador.

Capítulo 7 Questionado (**Frequentemente**

perguntado)

Os leitores podem ser curioso sobre como essas ideias surgiram, comparar com os outros na indústria, o quão longe eles podem ser esticados, eo que fazer quando eles não parecem aplicar-se. O capítulo é apresentado em forma de perguntas e respostas para permitir a "falar" as ideias, tudo a partir de fundamentos filosóficos de "como faço para começar?"

Crystal Clear funciona porque o desenvolvimento de software é um jogo cooperativo economicamente restrito da invenção e da comunicação e as regras do Crystal Clear melhorar os aspectos de invenção e de cooperação de trabalho da equipas.

Paradox que possa parecer, é precisamente a falta de requisitos e documentos de planificação *junto com* tendo os membros da equipas na mesma sala que melhorar as chances de sucesso. Os membros da equipas obter um feedback rápido sobre o processo e o produto, e ter apenas uma curta distância entre eles e as pessoas que precisam de saber a informação. Uma pessoa que aprende algo pode passá-lo para os outros com muito pouco custo de comunicação. A pessoa que necessita de saber algo pode obter uma resposta com semelhante pouco custo das comunicações. Ao tornar mais fácil para coordenar, com menos coordenar, o time reduz o custo e erro.

Neste capítulo, eu começo desde o início: a pesquisa que deu origem às recomendações, a ideia subjacente de desenvolvimento de software de ser um jogo económico-cooperativo, e a herança filosófica da visão de mundo incorporada. Essas conversas são bastante pesado e não para todos os leitores. Como uma pessoa colocá-lo, Crystal é "curto em práticas e longo em princípios." 48 Essas perguntas me permitir falar sobre os princípios. Após os princípios, eu rever as condições de contorno de Crystal Clear, distintas de Crystal Orange, por exemplo, e a falta de técnicas necessárias. I discutir a estrutura de metodologias em geral, e como isso influencia a estrutura deste livro e Crystal Clear em geral. I incluem um gráfico de resumo para Crystal Clear.

Após os primeiros quatro, perguntas pesadas, discuto a maneira em que este livro é apresentado, como o Crystal compara com Scrum, XP, e com a norma ISO 9001 e CMM requisitos (I), distribuído e equipas maiores. Finalmente, a última pergunta: "Como faço para começar?"

48 Graças a Andy Pols do Reino Unido.

Questão 1. Qual é o fundamento para Crystal?

A primeira e melhor aterramento é empírica. A evidência empírica começou a aparecer depois que eu foi empregado em 1991 pelo então incipiente IBM Consulting Group para criar uma metodologia para as suas próximas projetos de tecnologia objeto. Eu não sabia nada de especial sobre a metodologia no momento, então meu chefe na época, Kathy Ulisse sugeriu que eu visitar e equipas de projeto debrief para descobrir o que tinham aprendido (obrigado, Kathy!). De lá para cá, visitei projetos tanto dentro IBM e fora IBM, em todas as partes do mundo. Eu continuo estas debriefings projeto até agora, porque o relatório de pessoas é tanto informativo e surpreendente.

O que eu aprendi foi que a maioria do que tinha sido escrito nos livros de metodologia foi bastante irrelevante para a vida no projeto e tinha muito pouco a ver com a probabilidade do projeto de sucesso. Na verdade, até a chegada do XP, a correlação foi invertida, que quanto mais ocupados do grupo foi com seguindo as restrições da metodologia, menos provável era que eles iriam entregar o software em tempo hábil.

Os projetos que entrevistei que tinham conseguido muitas vezes tinham um processo desleixada aparência. Estas pessoas bem sucedidas falou sobre temas que não estavam relacionados com o método, mas Comunicação-e entrega-relacionada: sentados juntos, ter acesso rápido aos utilizadores experientes, oferecendo com frequência e receber feedback rapidamente.

Inicialmente, eu derrubei todas estas notas, e não sabia o que fazer com eles -depois de tudo, eu tinha sido acusado de escrever para baixo uma "metodologia", que deveria consistir de papéis, produtos de trabalho, normas de marco e do gostar. Foi só depois de vários anos de entrevistas, três ou quatro tentativas para escrever uma metodologia útil, e trabalhando diretamente em projetos que a mensagem começou a afundar-se adequadamente: Aqueles realmente são as propriedades fundamentais de sucesso; as técnicas e detalhes do produto Work- da "metodologia" são realmente um segundo fator de sucesso ordem. Receba as pessoas juntas, comunicando-se com boa vontade, entregando muitas vezes e obter feedback rápido dos utilizadores, e a equipas provavelmente irá resolver o resto por conta própria, utilizando qualquer tecnologia e técnicas que sabem.

Olhando para trás, minhas notas de dez anos de entrevistas, essas mesmas recomendações à tona uma e outra vez. Eu lentamente reconheceu que eles formam uma metodologia, que contém algumas regras específicas, mas permite que as pessoas a agir como profissionais educados em seus campos e trabalhar o melhor caminho para o sucesso por conta própria. O problema que tem me ocupado durante os anos desde que a realização foi apenas a forma de escrevê-lo de uma forma que outra equipas poderia seguir, como identificar quais peças são mais críticos do que outros, e como para descrever as variações permitidas. Meu melhor palpite atual é este livro.

O segundo aterramento consiste de um conjunto de princípios destilados de assistir projetos de software de forma sócio-antropólogo, da leitura de literatura sobre comunicação humana e design processo. Eu descrevi estes princípios em comprimento em *Ágil*

Desenvolvimento de software (**Cockburn 2002 ???), em um par de artigos chamada "Aprendendo com a Agile Software Development" (Cockburn 2002a, b ??) e na minha tese de doutorado, "Pessoas e Metodologias em Desenvolvimento de Software" (Cockburn 2003a ???). Eu lista e resumir os princípios aqui:**

1. Diferentes projetos precisam de diferentes metodologia trade-offs. Isso deve ser completamente óbvio. A julgar pelas iniciativas em curso em grandes empresas ao redor do mundo para criar e padronizar uma metodologia única para todos os seus projetos, no entanto, não é evidentemente óbvio. Em "Seleccionar Metodologia do Projeto" (Cockburn 1999), propus dois eixos para a seleção: o número de pessoas que necessitam de ser coordenada, e do grau de dano que pode ser causado pelo mau funcionamento do sistema. Estes dois eixos explicar por que Cristal não é uma metodologia única, mas uma família de metodologias, e servir de base para a seleção de um determinado metodologia de cristal para começar. Dos dois eixos, considero o número de pessoas sendo coordenado como o mais significativo para começar com; a equipas muitas vezes pode moldar a metodologia para o outro eixo.

2. equipas maiores precisam de mais elementos de comunicação. tamanho da equipas é a crítica elemento distintivo entre os membros da família de Cristal. Crystal Clear, por exemplo, só é recomendado para equipas que podem alcançar Osmótico Comunicação. Uma vez que a equipas cresce para mais de uma dúzia de pessoas, ou sentado mais de 30 segundos a pé uns dos outros diferentes modos, de comunicação e coordenação são necessários.

3. Projetos relacionados com maior dano potencial precisa mais elementos de validação. UMA pequena equipas de seis ou menos pessoas programação do movimento das hastes de boro em uma reação nuclear devem, espero, usar mais cuidado no seu trabalho do que os mesmos seis pessoas programação um sistema de tempo livre, talvez para organizar sua coleção da receita, manter o controle de pontuações do bairro equipa de futebol, ou pedir comida para o trabalho tarde da noite. A diferença está na dimensão verificação e validação, não na dimensão de coordenação e comunicação. Para o reator nuclear, não deve haver comentários públicos e walk-throughs dos requisitos, o design, o código e os testes. . . deve ser manifesta e publicamente claro que o sistema funciona. Essa quantidade de verificação é improvável que valer a pena o tempo, energia e custo monetário sobre os projetos em tempo livre. O Crystal Clear não abordar nativamente o nível crítico de vida de danos.

4. Um pouco metodologia faz um monte de bom; depois disso, o peso é dispendioso. Não muito trabalho tem sido feito sobre o aspecto retornos decrescentes de metodologias. A notícia surpreendente é que um pouco vai um longo caminho, e que os retornos decrescentes definir muito rapidamente. Isto é surpreendente porque parece haver uma resposta reflexa que "mais metodologia é melhor", o que significa que mais processos, mais documentos, mais

relatórios de status irá melhorar a probabilidade de entrega atempada. Minhas entrevistas indicam o contrário, que menos é geralmente melhor, enquanto um cobre as lacunas com comunicação pessoal. Jim Highsmith dá o conselho para começar com menos do que você pensa que você precisa ... que é provavelmente tudo que você precisa; é mais fácil de adicionar um pouco de como você vá junto de remover como você vá junto. Esse conselho é incorporada Crystal Clear.

5. Formalidade, processo e documentação não são substitutos para a disciplina, habilidade,

e compreensão. Este princípio de Jim Highsmith (2003) articula o que é diferente entre projetos ágeis e tradicionalmente preditivos ou conduzido com o plano. Software é construído por pessoas que precisam de inventar e comunicar as suas ideias, com base em sua habilidade e compreensão. As grandes corporações muitas vezes cometem o erro de pensar que ter pessoas preencher formulários, siga os passos específicos no desenvolvimento e documentar o estado atual de seu código, irá conferir-lhes habilidade em inventar soluções. No entanto, os três elementos que Jim destaca, disciplina, habilidade e compreensão, são todos *interno* Propriedades de uma pessoa e não pode ser substituído por as formas exteriores.

6. Interactive, face-a-face comunicação é o canal mais barato e mais rápido para

Trocando informações. Duas ou três pessoas que estavam em um quadro branco, diagramação e falando, são capazes de tirar vantagem de uma variedade de canais de comunicação e obter feedback quase instantâneo. Eles podem ver as expressões uns dos outros e frases de mudança em meados dos enunciados. Eles podem se mover mais perto e mais longe, ponto, gesto, fazer movimentos faciais, desenhar e falar todos ao mesmo tempo, interrompendo um ao outro e adicionando ao desenho como eles vão. Este tipo de situação cambial é conhecido como *caloroso,* o que significa que ele é rico em informações e também em emoção. À medida que as pessoas se deslocam para telefone, e-mail, e, eventualmente, de papel, a riqueza, a velocidade de interação e conteúdo emocional diminuir, e a comunicação é conhecido como

***resfriador.* Embora existam algumas vantagens em usar os canais de comunicação mais frias (assincronia e isolar reações sociais-emocional são dois deles), há uma tremenda perda de eficiência de comunicação, que em um projeto de software traduz em perda de tempo e aumento de erros. Como a maioria dos projetos são tempo e custar sensível, Crystal Clear maximiza a taxa de progresso para o tempo e dinheiro gasto, mantendo as comunicações perto e rica (Osmo Comunicação tic)**

7. Aumentar feedback e comunicação reduz a necessidade de intermediário

entregáveis. Muitas metodologias estão cheios do que chamo de "notas promissórias". Estes são documentos que prometem que a equipas *vai* construir tal e por isso, que *vai* ser construído como um tal e tal tempo e *vai* parecido com isto ou aquilo. Estas promessas são primários necessários porque há um tal atraso de tempo entre quando a promessa é feita e quando o software é entregue. Com tal atraso de tempo longo, é claro que é natural que os patrocinadores, examinadores e os utilizadores querem saber que o progresso

está sendo feito eo que o sistema final será semelhante. O problema com estas notas promissórias é que muitas vezes os planos puros e os projetos não funcionam como prometido. Se, ao contrário, a equipas constrói alguma coisa e mostra para os utilizadores a cada mês, o atraso tempo é curto o suficiente para que eles não têm de fazer promessas elaborados - eles simplesmente mostrar os resultados do mês e aprender diretamente e corretamente o que é certo e o que está errado. desenvolvimento de software ainda é lento o suficiente e caro o suficiente para que algumas notas promissórias são necessários - o plano de entrega e casos de uso, sendo dois - mas o princípio ainda é que quanto menor o tempo de atraso, são necessários os menos, notas intermediárias "promissórias".

8. custo de desenvolvimento de câmbio desenvolvimento simultâneo e de série para a velocidade e flexibilidade. Este princípio diz que o desenvolvimento concomitante executado corretamente pode acelerar o desenvolvimento, a um custo de salário, possivelmente mais elevada, em comparação com o desenvolvimento de série corretamente executada, em que cada tarefa é executada até a conclusão antes da próxima tarefa é iniciada 49. O problema com o desenvolvimento de série de baixo custo é que qualquer erro na compreensão provoca um efeito cascata de retrabalho, que é caro. Ela se baseia em ter requisitos muito estável e uma compreensão completa e estável do domínio e da tecnologia de implementação. É muito raro em projetos de software que os requisitos e a tecnologia são suficientemente bem compreendida para permitir que essas eficiências de ser colhidas. A família de Cristal é, portanto, construída sobre desenvolvimento simultâneo, o que permite a descoberta em tempo real das suposições equivocadas, mas, ao mesmo tempo, exige uma melhor comunicação entre as pessoas (que recebe de volta a Fechar Comunicação e Comunicação osmótica).

9. Eficiência é dispensável em atividades não-gargalo. O livro, *O objetivo,* por Elihu Goldratt, identifica que cada processo tem uma estação de "gargalo", que restringe a velocidade de toda a empresa. desenvolvimento de software não é diferente. O que o livro não recorrer, mas a família de cristal faz, é o corolário, que as estações não-gargalo pode ajudar em determinadas maneiras, operando com eficiência inferiores. Eficiência torna-se uma mercadoria dispensável. No desenvolvimento de software isso se manifesta quando as pessoas com ajuda capacidade ociosa para escrever requisitos, testes ou documentação; ou, quando os desenvolvedores (assumindo que eles têm capacidade de reposição) começar a trabalhar antes que os requisitos são finalizados, tendo sobre si a probabilidade de retrabalho mais tarde. Exemplos destas situações são descritos em *Desenvolvimento Ágil de Software.* Identificar a estação gargalo e fazendo uso da capacidade ociosa deve tornar-se uma parte essencial de oficinas metodologia de modelagem da equipas.

---

49 O terceiro capítulo da *Gestão simultânea (* Laufer 1998) descreve como a malha desenvolvimento simultâneo e de série no mesmo projeto para tirar proveito de cada um. Tão interessante quanto as técnicas que ele descried é o fato de que suas histórias vêm de projetos de engenharia civil (por exemplo, a construção de um hospital, ou uma pista de pouso na selva).

10." sweet spots" de desenvolvimento de velocidade. O melhor de todos os mundos possíveis é ter (1) dedicado, (2) as pessoas experientes que (3) sentar-se ao alcance da voz do outro, (4) usar testes de regressão automatizados, (5) têm fácil acesso aos utilizadores; e (6) entregar em execução, sistemas testados para esses utilizadores a cada mês ou dois. Tal projeto está claramente em uma posição melhor para concluir com êxito de uma falta dessas características (é surpreendente que os executivos que patrocinam não prestar mais atenção a esses fatores importantes de sucesso).

Crystal Clear é construído em cima e legisla o último quatro desses pontos doces. Obviamente, é uma equipas raro que pode ser composta por pessoas experientes, e assim por Crystal Clear trabalha com a hipótese de que o projeto é composta por uma mistura de desenvolvedores experientes e novatos (embora exija que o *Designer-chefe* é experiente). Quando a equipas não pode bater um daqueles pontos doces, então eles precisam inventar uma maneira de se aproximar dela. o Foco propriedade é sobre permitir que a equipas para chegar perto do primeiro doce, mesmo quando eles não são totalmente dedicada a apenas um projeto. A questão do que fazer quando alguns dos outros pontos doces são perdidas é retomada em questões posteriores.

Uma vez que é difícil lembrar de dez princípios, eu derivar-los de uma ideia, a do jogo económico-cooperativo (descrito em pormenor no meu site 50, no *Agile Software Development,* e em "The End Of Engenharia de Software eo início da Económico- Cooperativa Gaming" (Cockburn 2004a).

O manifesto jogo cooperativo diz:

> **"O desenvolvimento de software é uma série de, meta-dirigida jogos cooperativos de recursos limitados de invenção e comunicação O objectivo primário de cada jogo, é a produção e implantação de um sistema de software;. O resíduo do jogo é um conjunto de marcadores para auxiliar o jogadores do jogo seguinte. as pessoas usam marcadores e adereços para lembrar, inspirar e informar-se mutuamente no sentido de obter para o próximo movimento no jogo. o próximo jogo é uma alteração do sistema ou a criação de um sistema vizinho. portanto, cada jogo tem como um objetivo secundário para criar uma posição vantajosa para o próximo jogo. uma vez que cada jogo é, os objetivos primários e secundários competir por recursos com recursos limitados ". A vantagem desta formulação é que ela destaca um par de aspectos extremamente importantes de desenvolvimento de software:**

- **É o *pessoas* e sua *invenção* e *comunicação* que o software fazer acontecer. Tudo o que se relaciona com a sua velocidade de invenção e de comunicação afeta o resultado do projeto. Isso inclui especificamente proximidade, comunidade, amicability, conflito e confiança, como tem sido discutido longamente neste livro.**

50 http://alistair.cockburn.us

- **Há dois objetivos em movimento a cada instante: Entregando o presente sistema e ajuste para o jogo seguinte. Eles conflito, exigindo uma combinação não-trivial de curto prazo "gananciosos" e de longo prazo ações "investimento".**
- **Cada decisão, por cada pessoa envolvida, tem consequências económicas. Dado que a situação é over-constrangido e com poucos recursos, isso normalmente significa que você vai doer, não importa o que você faz; só se escolher de que forma se machucar. Se você gastar muito tempo em ações de investimento, você se machucar no curto prazo (e talvez a longo prazo); se você faz muitas ações gananciosas, você se machucar no longo prazo; e provavelmente não há terreno seguro meio. Não adormeça. Eu gerar a maioria das minhas recomendações metodologia e estratégias de gestão de projeto do manifesto jogo cooperativo. Considerando o estado do projeto, os dois gols, e a natureza da comunicação homem-a-homem, eu sou capaz de escolher aonde para abreviar, aonde gastar esforço extra, quando colocar as pessoas mais próximas, e quando colocá-los ainda mais além. (Sim,**

***reduzir* comunicação! Consulte "O Cone do Silêncio" (Cockburn 2004b).)**

* * *

O pesquisador sueco Pelle Ehn e desenvolvedor dinamarquês Peter Naur fornecer aterramento adicional para essas ideias com base nos filósofos Descartes, Marx, Heidegger, Wittgenstein e Ryle. Ehn examina os quatro primeiros destes filósofos em sua excelente, mas infelizmente fora de catálogo *Construção orientada para o trabalho de Artefatos de computador 51.* Eu resumir sua extensa discussão de uma forma breve e necessariamente grosseiro da seguinte forma:

- **Descartes nos dá uma visão de mundo que há uma realidade objetiva que pode ser descrito. Esta visão de mundo subjacente e informa desenvolvimento dominante software tradicional: A atribuição da equipas de desenvolvimento é capturar essa realidade em suas exigências, no seu código, e depois novamente em seus modelos de análise, design e documentação.**

- **Wittgenstein nos dá uma visão de mundo contrária, que não há nenhuma realidade objetiva que podemos articular e acordar. Pelo contrário, estamos todos empenhados em curso "jogos de linguagem" que evoluem ao longo do tempo e que exteriorizam uma parte pobre da nossa compreensão a qualquer momento no tempo. A equipas de desenvolvimento não "captura" as exigências, porque nem os requisitos nem entendimentos das pessoas são ou exprimível ou compartilhado. O que acontece é uma espécie de dança em que as várias pessoas executar ações e fazer declarações, e reagir às ações e declarações de todos os outros e também a si mesmos. Embora esta descrição pode parecer desarrumado e vago, que corresponde ao meu entendimento particular do que**

51 Um extrato de escrita de Ehn está incluído na *Apêndice B* do *Desenvolvimento Ágil de Software.*

acontece em projetos, e constitui a mais profunda base subjacente para a família Cristal e Crystal Clear mais particularmente.

- **Marx nos dá a visão de mundo que cada novo sistema de software irá mudar a estrutura de poder social. Assim, os utilizadores, o executivo e os desenvolvedores devem incluir em seu processo de desenvolvimento de um inquérito sobre como o sistema irá transferir o poder, e como lidar com essa mudança. Esta visão de mundo**

  é levado em conta em pequena escala pelas metodologias ágeis, que dão uma voz maior para os utilizadores. Foi tomado deliberadamente em conta por Kristin Nygaard, Pelle Ehn e a escola escandinava de desenvolvimento na década de 1970 e 1980, quando eles estavam trabalhando com os sindicatos locais sobre aonde e como incorporar a tecnologia de computador no local de trabalho sindicalizada.

- **Heidegger discutiu a natureza de ferramentas, como sendo ou visíveis para o portador como um objeto ferramenta, ou invisível para o portador, simplesmente uma extensão do seu corpo que realiza a tarefa necessária. Enquanto a ferramenta está visível para o portador ("Gee, este martelo é pesado", ou, "Aonde está a chave de hifenização, de novo?"), A ferramenta interfere com o desempenho da tarefa. Quando o portador está realizando a tarefa a toda a velocidade, ele não percebe sua existência em tudo. Essa discussão é importante para os designers de interface do utilizador e também do modelo de domínio, o que pode interferir com a usabilidade do sistema.**

- **visualizações de Ryle são discutidos por Peter Naur em "Programação Como Theory Building" (Naur 1990). Naur incorpora a ideia de Ryle de "teoria" para descrever a maneira pela qual um designer-programador tem de compreender o domínio em mãos, os requisitos, a tecnologia e a solução de projeto proposto. Para um designer para fazer alterações simpáticas exige que ele tem um entendimento rico o suficiente para ser extraído de implicações e causas. Para um grupo de pessoas para tomar decisões de design harmonioso, eles têm que ter em suas cabeças separadas, teorias que são "fechar" um ao outro. Para mim, isso explica muito ordenadamente o elemento metáfora do XP: a metáfora é uma frase curta que aponta para a teoria do design do sistema; acordo, o melhor da equipas na metáfora, *Agile Software Development,* pp. xxxx). A ideia de Naur de cada pessoa que tem sua própria teoria do sistema, e a dificuldade de alinhar as várias teorias em toda a equipas de desenvolvimento se encaixam naturalmente com a noção de Ehn e Wittgenstein de jogos de linguagem e a dança de ações e declarações que se movem as pessoas a um maior alinhamento da linguagem e compreensão.**

Ehn discute a idéia de pessoas que estabelecem marcadores de seus entendimentos atuais, para que eles possam consultá-los mais tarde. A escrita de Naur leva a reconhecer que a sutileza de entendimentos das pessoas mudam ao longo do tempo, e que o design atual, bonito ou deselegante, realmente é um espelho da sua atual

compreensão 52. Como sua teoria amadurece, torna-se mais sutil, e que recebem uma expressão design mais concisa e natural.

Há um lado negativo de ter uma teoria mais sutil. O projeto torna-se mais sutil ao longo do tempo, o que significa que o mantenedor também precisa desenvolver uma teoria semelhante sutil para assumir o código! Realmente desenvolvedores experientes encontrar este frustrante, porque isso significa que o seu melhor, o código mais limpo, mais concisa é praticamente insustentável! Eles assistem com horror como as pessoas que os seguem descompactar o projeto para se tornar maior, desajeitado, e *mais evidente para o observador casual!*

Essa observação põe em causa a própria ideia de qualidade no design. Uma das muitas dimensões da qualidade é de manutenção e capacidade de evoluir. Para o desenvolvedor mais experiente, o design sutil é "melhor". Ele pode, com sua compreensão sutil, ajustar o sistema às novas circunstâncias, com apenas a menor das mudanças. Isso mesmo código é quase insustentável para a pessoa com uma teoria menos desenvolvidas. Com o design menos sutil, a segunda pessoa vai demorar mais tempo e tem que fazer mudanças mais extensivas, e ao mesmo tempo encontrar que o design menos sutil "mais fácil" para manter e estender. Eu não vejo nenhuma maneira de sair deste dilema; é apenas mais uma daquelas situações contraditórias que infestam desenvolvimento de software.

Experiencial, fundamentos cognitivos e filosóficas sugerem que a melhor maneira de desenvolver software é usar tão poucas pessoas que tudo pode ser comunicados em discussões cara a cara, deixando pequenos grupos finesse problemas confiando no bom senso e habilidade; manter os comportamentos e artefatos juntos adotando apenas alguns princípios do desenvolvimento, tais como o envolvimento do utilizador direta e rastreamento por produtos acabados. Estes são os fundamentos da Crystal Clear.

Os leitores que se preocupam com essas coisas devem examinar Ehn e os escritos de Naur para entender melhor os fundamentos filosóficos e cognitivos da abordagem de Cristal.

*** * ***

Há um viés na maior parte não declarada em Crystal, a das pessoas geralmente confiantes. Essa tendência vem de mim. bagagem psicológica inevitavelmente vaza de um autor metodologia em suas recomendações, e este é o meu.

Não é uma confiança cega, no entanto. Embora equipas de melhor desempenho e usar menos energia quando eles podem trabalhar em confiança, nem todas as pessoas são capazes de confiar nos outros, nem todas as pessoas merecem confiança. Algumas pessoas, jogando seu próprio jogo de sobrevivência de carreira, são bastante dispostos a tirar proveito de seus companheiros de equipas ou sub otimizar-os resultados do projeto se ele vai melhorar suas carreiras. Outros não são capazes de cumprir as suas atribuições. Ainda outros simplesmente não se importam. Em geral, cada pessoa está observando todos os outros para ver

---

52 Ward Cunningham disse palavras quase idênticas para mim em 1994. Eu não poderia ter notado que nos escritos Ehn e Naur de se Ward já não tivesse mencionado isso.

quando a confiança vai falhar. Esta é uma razão que a propriedade específica de Crystal Clear é o mais fraco Segurança pessoal, e não completa Confiar em.

O gerente de uma equipa "jogos cooperativos" deve perceber e atender às falhas na confiança. Um dilema que mostra periodicamente-se é a seguinte: O *Patrocinador executivo* pede aos programadores quanto tempo um conjunto de tarefas irá tomar. Eles respondem, o *Patrocinador executivo* aceita, e os programadores começar a trabalhar. Com o passar do tempo, o *Patrocinador executivo* recebe essa sensação desconfortável de que os programadores estão a tirar partido dela. Será que a atribuição realmente levar tanto tempo (os programadores estão trabalhando diligentemente), ou eles estão tirando proveito da situação e preenchimento suas estimativas de tempo para que eles possam relaxar junto a um ritmo lento arbitrariamente?

Não há maneira de resolver a questão, seja em cristal ou de qualquer outra metodologia. Ou poderia ser verdade e eu encontrei o que eu acredito que são exemplos de ambos.

desenvolvimento de software é um jogo de pessoas que trabalham com e contra outras pessoas. Todos os motivos e emoções humanas aplicar e deve levar em conta, mesmo em Crystal.

Questão 2. Qual é a família Crystal?

Crystal é uma abordagem ao desenvolvimento de software baseado em um código genético. Esse código genético permite que instâncias específicas de cristal a ser gerado para diferentes circunstâncias; todas as instâncias irão compartilhar uma semelhança familiar.

O código genético é constituído por

- **o modelo de jogo económico-cooperativo que acabamos de descrever,**
- **um conjunto de *prioridades* e *princípios* para fazer escolhas,**
- **selecionado *Propriedades* para dirigir para,**
- **amostra *estratégias* e *técnicas,***
- **amostra *casos* copiar. As três prioridades são: projeto *segurança,* desenvolvimento *eficiência,* *habitabilidade* das convenções resultantes.**

o *segurança* prioridade é obter um resultado de negócio razoável a partir do projeto, dadas as prioridades do projeto e as restrições de recursos. *Eficiência* em desenvolvimento é uma prioridade porque muitos projetos são economicamente overconstrained.

***habitabilidade* tornou-se uma prioridade quando comecei a notar nas minhas entrevistas projeto que a maioria dos programadores têm o poder de ignorar tudo o que a metodologia está mandatada pela sua organização. Todos eles têm de fazer é dizer que ele diminui-los e eles vão perder seu prazo se eles fazem todo esse trabalho extra. Que normalmente é base suficiente para ignorá-lo. Para ser justo, a maioria das metodologias são tão ineficiente que eles estão corretos em sua declaração, por isso não está claro se eles estão fazendo a organização um desserviço ou um favor ao ignorar a metodologia (minha estimativa é que há um imediato 15% -30 % de eficiência a ser adquirida na maioria das organizações simplesmente jogar fora a maior parte de seu processo 53). Se os programadores dizem que eles tem que fazer isso de qualquer maneira, eles costumam encontrar maneiras de contornar isso. Não importa o que a metodologia é, se ele não se acostumar. Em outras palavras, *habitabilidade* é um fator crítico de sucesso da adoção da metodologia.**

Parte da habitabilidade é a tolerância para a variação humana. Alguns gostam de fazer listas, outras não; algum trabalho melhor visualmente, outros com texto; alguns gostam de trabalhar sozinhos, outros em grupos. Alguns olhar para o desenho por horas, dias ou semanas ter certeza que ele está certo, enquanto outros gostam de "sentir" o código tomando forma. É por esta razão que Crystal é tem muito a tolerância em torno técnicas - não só eles mudam o tempo todo, mas as pessoas diferentes são atraídos para diferentes técnicas. o *habitabilidade* prioridade significa que as regras precisam ser tal que as pessoas da equipas podem viver com eles.

As prioridades interagir: *habitabilidade* significa que eu aceito que um conjunto escolhido de regras pode não ser tão *eficiente* que possível. o *segurança* prioridade significa que as regras escolhidas devem ser

---

53 Como aponta Luke Hohmann out: "Quando você quiser que o seu barco para ir mais rápido, é mais fácil de cortar âncoras do que adicionar cavalos

de potência."

bom o suficiente para obter um resultado adequado (fazendo exceção, é claro, para gerentes de projeto pobres e patrocinadores gananciosos, os quais ainda existem).

Parte da eficiência e habitabilidade está lidando com o fato de que cada projeto é ligeiramente diferente, e assim corretamente precisa de sua própria metodologia, adaptada.

Isso significa que ninguém metodologia será suficiente para a gama de projetos que mesmo uma empresa de médio porte vai encontrar.

A empresa não precisa definir dezenas de metodologias, no entanto. Uma empresa com amplamente diversos projetos foi capaz de fugir com apenas três metodologias de base: um para projetos sub-quatro pessoas, outro para projetos de oito a vinte pessoas, e um terceiro para os sistemas que têm de passar os EUA Food and Drug certificação processo de administração. Cada equipas adapta a metodologia base mais próxima a determinados requisitos, experiência e tecnologia características do seu projeto.

Claro, eles têm de fazer este ajuste rapidamente o suficiente para caber dentro do horizonte de negócios-valor de custo. Se metodologia de modelagem leva a metade do orçamento do projeto, em seguida, seria de fato melhor usar uma

metodologia abaixo do ideal, mas padrão. A necessidade de eficiente shaping metodologia é por isso que eu descrever o *metodologia Shaping* e *workshop de reflexão* técnicas neste livro.

Essas são as prioridades. As propriedades são apenas aqueles descritos no capítulo 2. Não há estratégias ou técnicas são necessárias para uma equipas, além de desenvolvimento incremental. O conjunto de estratégias e técnicas permitidas é muito grande, e assim eu apresento apenas alguns dos mais interessantes e menos conhecidos no Capítulo 3. amostras Metodologia estão incluídos no *Sobrevivendo Projetos OO (* Cockurn 1998), *Desenvolvimento Ágil de Software*

(Cockburn 2002), e, claro, este livro.

Para ajudar com a escolha de um conjunto de regras para a um projeto particular, eu construí a grade bidimensional mostrado na Figura 7-1.

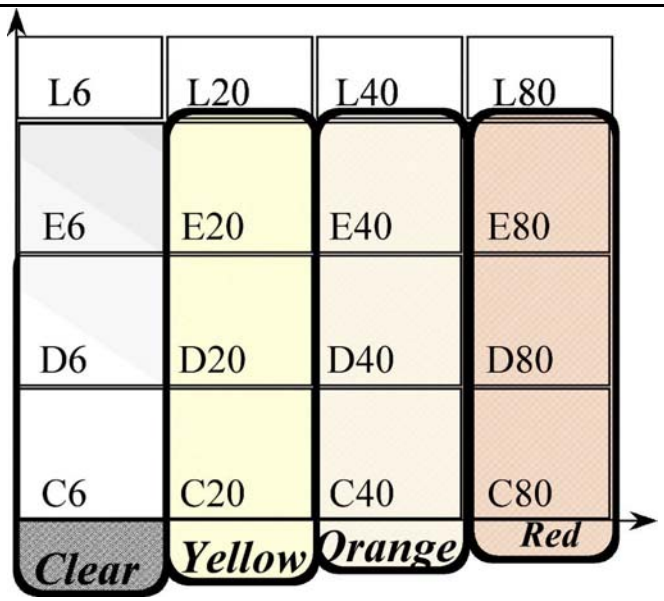


Figura 7-1. cobertura de cristal de diferentes tipos de projetos. esta grade 54 identifica *tamanho da equipas,* o número de pessoas a ser coordenado, como o eixo primário no que diz respeito a metodologias de definição. Uma segunda dimensão é importante *criticamente,* o dano potencial causado por um defeito sem ser detectado: perda de conforto (C), a perda de quantias discricionários (D), a perda de quantias essenciais (E), e perda de vida (G). Devemos esperar para ver os componentes de comunicação e coordenação de metodologias mudar visivelmente como a equipas mantém dobrando de tamanho. Devemos esperar para ver os componentes de validação e verificação alterar visivelmente como a criticidade aumenta até perda de vidas (aonde o processo de aprovação da FDA podem tornar-se relevante). Na Figura 7-1, a caixa C6 significa projetos com até seis pessoas trabalhando em um sistema de perda de conforto.

O nome *Cristal* deriva estas duas dimensões, em analogia com cristais geológicas, que são também caracterizadas em duas dimensões: cor e arnês. Eu deixei o tamanho da equipas (mais geralmente, a comunicação e coordenação complexidade) correspondem à escuridão da cor: claro, amarelo, laranja, vermelho, marrom, violeta azul e assim por diante, e a criticidade correspondem ao arnês mineral: soft como quartzo para sistemas de conforto perda de de-, duro como diamante para sistemas de perda de vida.

A Figura 7-1 mostra caixas de 16, o que implica a necessidade de pelo menos 16 metodologias para ser definido. A minha experiência é que, se um grupo começa a partir de uma metodologia documentado e formas na mosca com o *Metodologia Oficina Shaping* e a *Workshop de reflexão,* as coisas podem simplificada. Uma sugestão é colocar a vida críticos de

54 Descrito em (Cockurn 1999ieee) e em *Desenvolvimento Ágil de Software.*

projeta em sua própria área, e agrupar o resto pelo tamanho da equipas. Isso deixa uma dimensão primária - cor - como a característica distintiva de metodologias não-crítica da vida. Eu não tenho experiência suficiente ainda para saber como o conjunto de metodologias críticas de vida, mas a empresa mencionada acima decidiu que todos os seus projetos de aprovação pela FDA com menos de 20 pessoas poderiam ser agrupados. Eles não tem nenhum projetos em todos com mais de 20 pessoas, para que eles simplesmente tinham três categorias, que podemos chamar *claro, amarelo,* e *diamante.*

Questão 3. Que tipo de descrição metodologia é isso?

A forma predominante de transmitir uma metodologia é através de um elemento descrição elemento-a-passo de, aproximadamente, uma dúzia de diferentes tipos de elementos. A Figura 7-2 mostra um conjunto genérico e típicos de tipos de elemento metodologia. Um pouco multiplicação mostra que mesmo uma metodologia bastante simples exige a descrição de mais de uma centena de elementos separados, além de como cada um se encaixa com todos os outros. A maioria das metodologias conter várias centenas de elementos, e Extreme Programming tem cerca de quatro dezenas. Você pode ver imediatamente por descrições metodologia são tão longo (e chato).

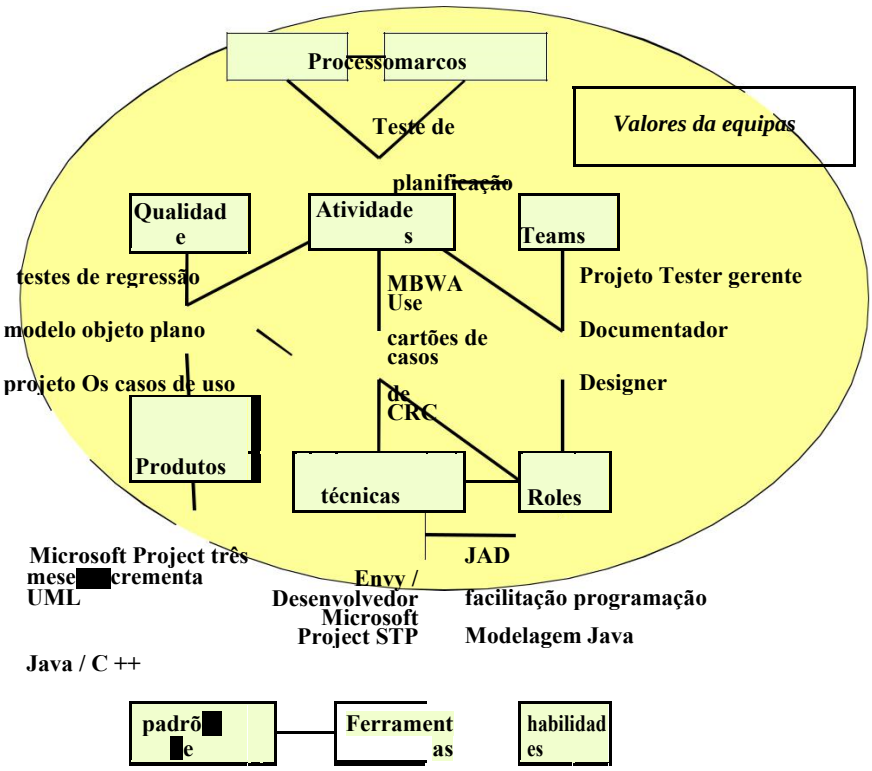


Figura 7-2. tipos de elementos de uma metodologia com elementos da amostra.

Parte do texto metodologia descreve a "fábrica" aonde o software é produzido (Eu não gosto das conotações históricas dessa palavra, mas não consegue encontrar outro). A parte não-fábrica da metodologia descreve como os produtos de trabalho devem ser descritas. A Figura 3/7 mostra a separação de elementos metodologia em "fábrica" e descritores de produtos (com o tempo que flui para baixo no diagrama).

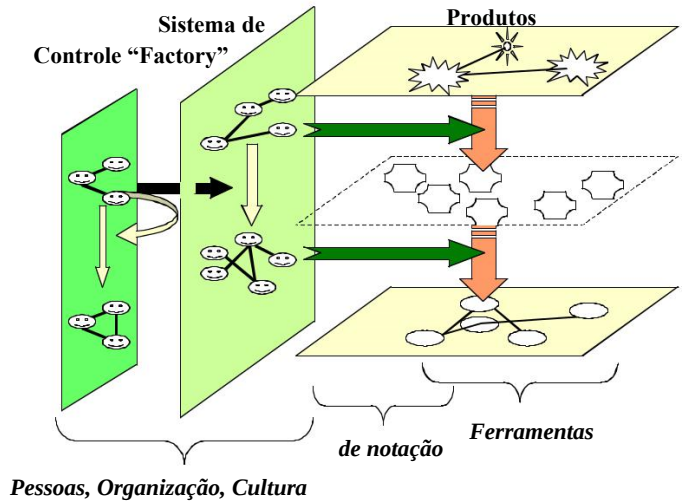


Figura 7-3. Divisão de metodologia em "fábrica" e descritores de produtos. À direita, vemos as descrições originais, distorcido e imprecisas do produto (Visão e Missão declaração, protótipos conceituais) no topo, evoluindo ao longo do tempo em produtos de trabalho intermediários, como requisitos e planos, e eventualmente em código-fonte, testes, embalagem, manuais de utilizador. Os elementos da metodologia cobrindo esta linha evolução no domínio do notações e técnicas. Assim, vemos que UML é um produto de descrição padrão de notação (que explica tanto por que é útil, e ainda afeta o resultado de projetos tão pouco).

As duas fatias verticais à esquerda são a "fábrica" e seu sistema de controle, ambos feitos de pessoas. As pessoas que produzem o software viver em uma estrutura organizacional, que muda ao longo do tempo. As pessoas que causam essas mudanças viver em sua própria estrutura organizacional, que também está mudando ao longo do tempo. Os elementos da metodologia que cobrem estas preocupação das pessoas, organização e cultura. Historicamente, grupos de desenvolvimento de software têm sido avessos a discussão explícita dos seus elementos organizacionais e culturais, que tem sido uma fonte contínua de disfunção em nosso campo.

As setas horizontais indicam as ferramentas que as pessoas usam para criar e alterar os produtos de trabalho.

Eu tenho usado estes quadros para descrever metodologias de anos. Duas falhas neles são, no entanto, lentamente se tornando tanto aparente e dolorosa.

Uma falha é a idealização de pessoas em papéis. Uma descrição metodologia é, por sua própria natureza, uma fórmula destinada a cobrir vários projetos, e a parte principal que fica abstraída é *personalidade.* O problema é que as pessoas estão simplesmente cheios de personalidade, de modo que o *pessoas* que aparecem no trabalho não têm o ajuste correto com o *papéis* que é suposto estar em funcionamento. Não é um "tester" que vem para trabalhar na parte da manhã, é Jim; ele não é um "gestor de projeto", é Annika; ele não é um "desenhador", é

Peter. Annika pode ou não ter a personalidade necessária para um bom gerente de projeto, Peter pode ou não ter o conjunto de habilidades necessárias para ser um bom designer (veja a Figura 7-4).

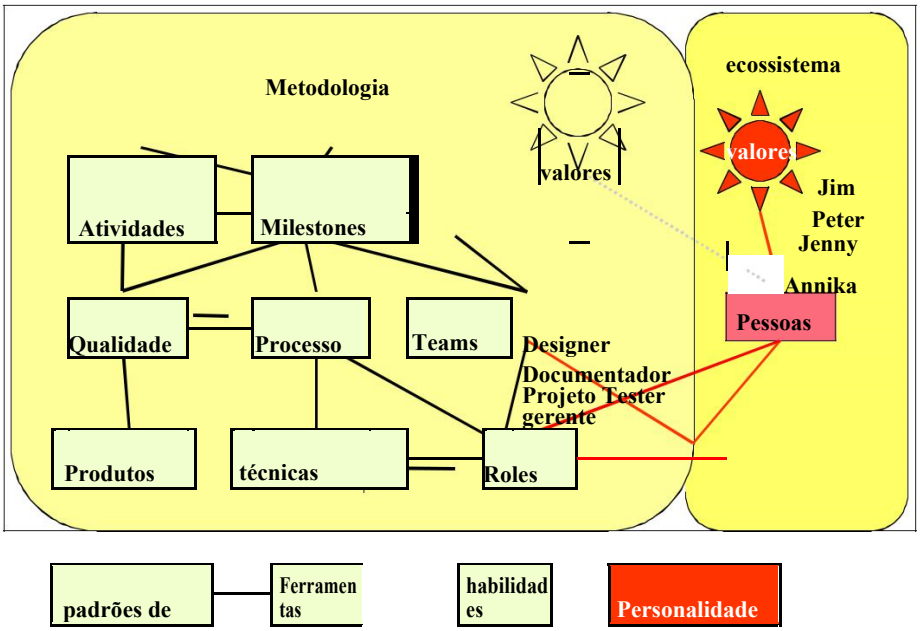


Figura 7-4. Metodologias idealizar personalidade, mas as pessoas são recheadas cheio dele.

Na prática, personalidades individuais dominar. Peter pode ficar ofendido por Annika e se recusam a entregar relatórios; Annika pode encontrar Jim difícil falar com e ignorá-lo durante semanas de cada vez; ou, como muitas vezes acontece, Peter pode simplesmente assumir o projeto, ignorar Annika e dirigir o projeto em qualquer direção que ele quer.

A segunda falha na apresentação padrão de metodologia é que há um número arbitrário de itens não classificados que não se encaixam no diagrama metodologia, mas acabam por ser importante para a forma como o projeto é executado. Eu encontrei como críticos itens "metodológicos": as formas das mesas, o layout das salas, a presença ou ausência de ventilação, uma tradição cultural de quinta-feira jogos de vôlei tarde ou bustos noite cerveja Sexta-feira, o estabelecimento de uma assinatura cultural como um logotipo, dizendo, ritual ou um valor como excelência, trabalhando até tarde, ou sempre sendo colocado de volta, e regras como "vendedores não são permitidas na área de programação, após 10:00" ou "sem telefonemas ou conferências entre 10 : 00 e 12:00 ".

Para resumir, eu encontrei quatro questões espinhosas nas descrições metodologia:

- Eles são longos, intrincado e chato;
- "Fábrica" descrição fica entrelaçada com a descrição do produto;
- Eles perdem muitas pequenas regras e questões que têm grande efeito;
- Eles perdem os efeitos de pessoas individuais com personalidades.

Eu decidi lidar com estas quatro questões da seguinte forma: Diferentes pessoas vêm para uma descrição metodologia de diferentes origens e capaz de perceber coisas diferentes (semelhantes aos cegos sentindo diferentes partes de um elefante e tentando formar sua impressão do todo). Por isso, escrevi os capítulos em diferentes formatos para dar pontos de vista diferentes para diferentes leitores de fundos. Eu não legislar técnicas, mas deixar aqueles com até os indivíduos como profissionais em seus respectivos campos. Os produtos de trabalho são apresentados através de amostras, para que eu não fique amarrado em padrões de notação que são susceptíveis de mudar a cada poucos anos. Como resultado Crystal Clear é principalmente uma descrição da "fábrica": aonde as pessoas se sentam, como eles interagem, e assim por diante.

Para cobrir as muitas pequenas regras que são importantes para projetos, eu experimentei em descrever as metodologias usando *Padrões* em vez da metodologia 12 tupla. Muito para o meu prazer, eu descobri que eu poderia descrever muitos aspectos de uma metodologia que eram importantes, incluindo as regras pouco estranho, de uma forma que foi facilmente compreendida e aplicada por equipas de projeto ocupados. Esses aspectos que não se encaixam como padrões poderia simplesmente ser expresso como convenções do projeto.

Fora dessas experiências eu aprendi que *uma metodologia é nada mais do que as convenções a equipas se compromete a adotar!* Estas convenções deriva ao longo do tempo, e assim não há nada mais natural do que a equipas deve reunir regularmente e rever suas convenções operacionais. Isto é, claro, o *Workshop de reflexão.* Além disso, as convenções não mais difícil é escrever para baixo do que frases simples, o que significa que eles podem ser escrito e publicado. Isto é mostrado no exemplo da *Convenções da equipas* produto de trabalho.

Estudar a literatura padrões, descobri que os bons padrões de cair em dois tipos: Propriedades e estratégias. Alguns são ambiguamente tanto, mas geralmente pode-se decidir que qualquer um é mais um do que o outro. Padrões fãs pode gostar de reconsiderar os padrões de Christopher Alexander sob essa luz. Por exemplo, *Luz em dois lados do quarto* é mais uma propriedade e *janela do sótão* mais uma estratégia (Alexander 1977 ???). O livro *Padrões de design (* Gamma 1995 ???) contém estratégias - que poderia muito bem ter sido chamado *Estratégias de design.* No *Sobrevivendo Projetos Orientados a Objetos,* I decidiu chamar a "Redução do Risco de apêndice *Estratégias"* em vez de "Padrões de redução de risco." Por outro lado, quando estávamos escrevendo *Padrões para Effective Use Cases,* deliberadamente procurou *Propriedades* de casos de uso eficaz e processos de casos de uso-escrita, ao invés de estratégias.

Quando chegou a hora de escrever este livro, eu tentei descrever a metodologia com padrões. No entanto, eles eram uma mistura perturbadora de propriedades e estratégias, Isso não é de todo um problema para qualquer equipas um projeto - eles só se preocupam com o que devem fazer-se. É problemático quando se aponta para um vasto conjunto de projetos, porque as estratégias devem ser as escolhas discricionárias da equipas. Portanto, eu separei os padrões em dois capítulos: Comunicação osmótica e Entrega Frequent entrou no capítulo propriedades e *Exploratórios 360 °, de passeio esqueleto* e *rearquitetura incremental* no capítulo estratégias. *workshop de reflexão* foi dividido fora como uma técnica, um meio para alcançar a propriedade Melhoria reflexiva.

A próxima questão envolve as idiossincrasias das pessoas específicas que aparecem nos edifícios específicos para trabalhar com tecnologias específicas sobre o problema específico na mão. Esta instância de pessoas, projetos e ambiente me refiro como o "ecossistema" do projeto (Cockburn 2002asd). Lidar com o ecossistema vem em duas etapas. O primeiro é reconhecer que as propriedades têm que ser o cerne da questão, e tudo o mais consultivo. Assim, o capítulo destilada capta as propriedades visíveis (seis a oito pessoas em uma sala ou salas adjacentes, etc.), mais as propriedades do grupo define no lugar (Entrega frequente, etc). Não menciona as estratégias, técnicas, ciclos de processos ou produtos de trabalho. Em vez disso, o grupo é necessário para obter periodicamente juntos e discutir em conjunto o que suas convenções locais deve ser. Quando alguém com uma personalidade forte ingresso ou saída, o grupo se reúne novamente para rever suas convenções.

A segunda parte de lidar com o ecossistema é reconhecer que a metodologia não pode resolver todos os problemas do grupo. Nenhuma metodologia pode corrigir a ausência de pessoas competentes e experientes, ou pessoas que não gostam uns dos outros, ou não pode tomar uma decisão. Estes são simplesmente exposições, e não apenas para Crystal Clear, mas a cada metodologia. É bem possível que um grupo, caso contrário, seguindo todas as recomendações de Crystal Clear, irá se reunir em sua *Workshop de reflexão,* têm um argumento e sair não ser capaz de trabalhar uns com os outros. Que está fora do alcance da metodologia.

O Crystal Clear pode fazer é nomear as regras que dão uma equipas de projeto melhor do que até mesmo chances de sucesso, e espaço de manobra.

Pergunta 4. O que é a folha de resumo para Crystal Clear?

tamanho do projeto	**Até 6 ou possivelmente 8 colocated desenvolvedores. Pode ser moldado com cuidado para até 12 pessoas.** **Não destinado a equipas maiores, uma vez que está faltando coordenação do grupo.**
o potencial do sistema para danos	**Perda de conforto ou perda de verbas discricionárias, como em sistemas de facturação. Pode ser moldado, com regras de teste, verificação e validação adicionais, até dinheiros "essenciais". Não se destina a sistemas críticos LIFE como ele está faltando a verificação da exatidão.**
Tipos de sistemas	**Mainframe, cliente-servidor, web-based, usando qualquer banco de dados tipo, central ou distribuída. Pode ser moldado para sistemas de tempo real rígido com regras adicionais para planificação e verificação de problemas de tempo sistema. Não se destina a prova de falhas de sistemas, uma vez que está faltando comentários duro de arquitectura para tolerância a falhas e fail-over.**
Os valores	**Strong em comunicações, luz sobre os resultados. , ricos, caminhos de comunicação informais curtas, inclusive com as comunidades patrocinadoras e de utilizador. entregas frequentes com o produto e feedback do processo. sobrecarga reduzida, e menos produtos intermédios de trabalho. Tolerância para variações de estilos de trabalho das pessoas.**
tolerâncias	**As normas de política são obrigatórias a menos que uma substituição equivalente pode ser feito. Por exemplo, o uso das técnicas de programação extremos para programação de projetos e estadiamento "Scrum" ou é aceitável; uso do lançamento e re-lançamento de técnicas de projeto TSP é aceitável.** **Cada entrega tem um proprietário ou proprietários designado. "Nenhum" e "Todos" são aceitáveis, se declarou explicitamente. Em cada um desses casos deve haver justificação adicional a respeito de como o modelo funciona (por exemplo, Extreme Programming declara que todos possui todo o código quando (e somente quando) eles trabalham em pares; programação em pares cria momento-a-momento dupla propriedade de código.**
Uso de precisão	**Baixa precisão no início de alta precisão, apenas em artefatos entregues. Baixa precisão inclui os designers falando com utilizadores e patrocinador, casos de uso de dois parágrafos ou histórias de utilizadores ou listas de recursos; desenhos no quadro branco, telas de utilizador desenhado como esboços a lápis, cronograma de liberação listas como curtas e frases (inicialmente).**

Agenda de Divulgação de listas curtas e frases (inicialmente). Alta precisão inclui demonstração e produção de telas, código final, casos de teste, manuais do utilizador ou texto de ajuda.

Pergunta 5. Por que os diferentes formatos capítulo?

livros de metodologia são geralmente escrito como um conjunto de regras dizendo ao leitor o que fazer. Existem várias razões para não usar esse formato neste livro. A primeira é que Crystal Clear não é tanto um conjunto de instruções sobre como escrever software como propriedades, condições, convenções e técnicas utilizadas regularmente para trazer uma certa classe de projetos para casa em segurança (este é o *segurança* prioridade). A intersecção desses projetos de sucesso é tão pequeno que não ser seguro em si, e a união de todos eles é muito gordo. Eu estou tentando, neste livro, para indicar como encontrar subconjuntos que são seguro e utilizável.

Os leitores vêm de várias origens, o que faz a diferença no que eles procuram no livro. Nossa prática de programação mudou drasticamente desde 1987 e com a chegada de objetos, e novamente desde 1998, quando o objeto e comunidades XP trouxe padrões, refatoração, a programação em pares e Test-Driven Development à tona. Alguns programadores têm mantido com as mais recentes mudanças e precisa ver referências apropriadas para essas práticas mais recentes. Outros ainda operam usando as práticas mais antigas, e precisam ser dada detalhe para apanhar com o mais recente vocabulário. Alguns leitores já especialistas que executam projetos de pequena equipas, estão à procura de novas ideias ou novas palavras para ideias antigas. Outros, novos para levar uma equipas pequena, quer instruções detalhadas. Aqui estão algumas das categorias de leitor que tentei lidar com:

- **Aqueles que chegam frio para o tema do desenvolvimento ágil pode se beneficiar a partir da discussão de e-mail mais leve e muito contextual no primeiro capítulo.**
- **Aqueles que já estão muito familiarizados com pequena equipas de desenvolvimento ágil pode querer se concentrar apenas sobre as propriedades de segurança do segundo capítulo e utilizar os seus próprios caminhos para chegar a essas propriedades.**

- **Aqueles começando com um Crystal Clear forma de trabalhar vai querer um conjunto inicial de técnicas. Algumas destas técnicas será novo mesmo para pessoas experientes, coisas que podem acrescentar ao seu conjunto de prática.**
- **Muito poucas pessoas que vêm de uma história tradicional processo de desenvolvimento de software têm uma compreensão operacional dos processos de desenvolvimento cíclicas. Aqueles com um foco do processo deve beneficiar de ver o processo desenrolou nas várias maneiras, e os ciclos isolado.**

- **Methodologists e proprietários de processos de desenvolvimento em geral, operam diretamente da lista de produtos de trabalho, avaliar uma metodologia baseada na lista de quem produz o que e quem verifica se. A recolha de amostras de trabalho ainda poderia dar um desenvolvedor ágil experiente algumas ideias novas.**

- **Aqueles que vêm para o livro com um conjunto variado de projetos na sua lista de ativos vai querer saber como variar as regras e quando mudar para uma metodologia completamente diferente.**

- **Aqueles que leram amplamente estarão interessados nos fundamentos económicos e filosóficos, e discussão de como essas ideias se encaixam com aqueles relacionados e concorrentes.**

Não é minha ambição que cada leitor deve gostar cada capítulo e formato. Em vez disso, a minha esperança é que através do uso de diferentes perspectivas, cada leitor, provenientes de seu fundo particular, pode encontrar algum formato capítulo que transmite o que eles precisam, a fim de compreender ideias principais do Crystal Clear.

Pergunta 6. Aonde está Crystal Clear no panteão dos metodologias?

Há muitas metodologias para comparar contra, por isso vou abordar esta questão, mostrando *quão* Eu responder-lhe e dar apenas dois exemplos.

Cada autor metodologia seleciona um par de prioridades a ser foco ou centro das atenções da metodologia. Esse foco poderia estar em *redução de defeitos,* como em salas limpas (ref ???) e método de programa-a derivação Dijkstra / Gries' (Gries 1983), *repetibilidade* como na norma ISO 9000 (ref ???), *previsibilidade e controle* como na CMM (I) (ref ???), ou *produtividade* como no XP. Já ouvi pessoas discutir como permanecer "tranquila" à medida que expandem, indicando que como uma prioridade. Seja qual for o autor prioriza torna-se um filtro para selecionar e rejeitar possíveis elementos da metodologia. Cuidado autores metodologia que afirmam que a sua concepção metodologia satisfaz todas as prioridades.

Ao comparar metodologias, os investigadores geralmente listam as técnicas e regras dos metodologias em uma tabela, e permite que os leitores avaliar a lista de acordo com as suas (os leitores) prioridades pessoais. Isto pressupõe que o leitor já entende as implicações de prioridades no projeto de metodologia. Se o leitor não entender, então é claro que ele pode ler a comparação e fazer uma escolha significativa. Se não, no entanto, (e a maioria dos leitores não sabem sobre as prioridades metodologia) há uma maior chance de que o leitor irá selecionar uma metodologia não alinhado com a equipas e conjunto prioridade real da organização.

Cristal, como uma família de metodologias, concentra-se em *eficiência* e *habitabilidade* como elementos críticos de *segurança do projeto.* Todas as metodologias de cristal tentar realizar, tanto quanto possível, com o mínimo possível. No entanto, essa unidade para a eficiência está sujeita ao imperativo habitabilidade. Equipas de todo o mundo têm diferentes origens e valores e são confrontados com projetos de diferentes características e prioridades. o

***habitabilidade* prioridade é dar a cada equipas e cada membro da equipas a escolha livre máximo em trabalhar o seu próprio caminho (queremos que a equipas não só para ter sucesso, mas também estar disposto a trabalhar de forma semelhante novamente).**

Obviamente, os dois conflitos em algum momento, e assim Cristal permite equipas para escolher formas menos-que-ideal de trabalhar se assim o desejarem. Enquanto eles manter a entrega de software com sucesso, estou satisfeito 55.

Crystal Clear é a versão do Crystal para pequenas equipas, colocated. Ele tem as mesmas prioridades e foco de atenção, ser eficiente e tolerante.

Como ponto de comparação, XP prioriza para programador *produtividade* e código *evolvability (* Atenção: estas são as minhas interpretações). XP reduz produtos de trabalho intermediários (aumento da produtividade), e levanta a disciplina necessária das pessoas (aumentando

55 Esta é, creio eu, uma diferença fundamental entre Cristal e XP.

produtividade de novo), mas reduz a tolerância para estilos de trabalho individuais. O código deve ser reformulado para se tornar mais fácil a evoluir, e há um extenso conjunto de testes de unidade para ajudar a próxima pessoa junto.

Os patrocinadores do projeto solicitando uma equipas de desenvolvimento ter ISO-9000 e CMM certificação (I) estão se concentrando principalmente na *previsibilidade* e *repetibilidade.* Note que não é a metodologia que será avaliada nos diferentes níveis de CMM (I), mas a própria organização. Para passar a certificação, a organização precisa de uma metodologia documentada, e que a metodologia deve ser projetado de forma a facilitar a obter a certificação.

CMM (I), em particular, carrega o conceito de que a variação entre os projetos podem ser reduzidos através de uma atenção adequada para o controle estatístico do processo. Para CMM certificação (I) Nível-4, a organização deve mostrar que ele mede o processo, e para a certificação nível 5 que usa as medidas para otimizar o processo. (Crystal transforma o I) pirâmide (CMM cabeça para baixo em um sentido, mas essa discussão pertence a uma pergunta mais tarde).

Eu não sei o suficiente para responder o que o foco de atenção é para o Rational Unified Process (Kruchten 1999 ???), DSDM (DSDM Consortium 2002 ???), ou Scrum (Scwaber 2002 ???).

Que prioridades dirigir a seleção de elementos de metodologia na sua organização? Quando você entende isso, você estará em uma posição muito melhor para decidir como escolher ou misturar outras metodologias.

Crystal Clear e XP

XP é semelhante ao Crystal Clear, em muitos aspectos, em particular com a sua atenção para colocation, iterações curtas, entrega frequente, e contato próximo com os patrocinadores e utilizadores finais. É mais rigorosas do que Crystal Clear de várias maneiras e mais solto em poucos 56.

- **ciclos de iteração do XP são obrigados a ser mais curto: um dia a um mês. XP não fazer quaisquer regras sobre quanto tempo o ciclo de entrega deve ser. Este é o reverso da Crystal Clear, que exige que as entregas de não mais de três meses na pior das hipóteses ser, mas permite que a iteração para ser tão longo quanto a entrega. Eu tenho visto de outra forma excelentes projetos XP entrar em apuros simplesmente porque o tempo de entrega era de seis meses a um ano. Esta é claramente uma violação da intenção de XP, que era *implicitamente* baseada em cima entregas frequentes. Para corrigir essas situações, Crystal Clear é explícito sobre a necessidade de entregas reais. Minha sensação é que a mudança de anual para entregas trimestrais exige a maior mudança mental na organização, e uma vez que a organização começa a entregas trimestrais, pode ver o propósito encurtando ambos os ciclos de iteração e entrega, e com as oficinas de reflexão, pode encontrar um conjunto que funciona para ele.**

56 Estes comentários são baseados na definição de XP 1998-2004. Kent Beck está ocupado mudando a definição de XP, assim que as respostas desta seção podem mudar ao longo do tempo. Use a mesma linha de raciocínio, como mostrado para derivar as respostas up-to-date como XP evolui.

- **XP requer programação em pares, que Crystal Clear não. A diferença é fundamental: XP pretende ser a metodologia mais eficaz possível, e, consequentemente, reduz os graus de liberdade da equipas em fazer esses tipos de escolhas. Crystal Clear pretende ser um tolerante e *adequadamente* metodologia eficaz, dando à equipas o máximo de liberdade possível na construção do conjunto local de convenções.**

- **XP requer Testes Unitários Automated com uma variedade de testes de unidade automatizados, e integrações múltiplas vezes por dia. O primeiro é apenas uma propriedade recomendada de Crystal Clear, e o período de integração é permitido variam significativamente entre os projetos.**

- **XP requer um cliente no local, aonde Crystal Clear só chama para fácil acesso a utilizadores experientes, com um mínimo indicado de uma hora por semana, em vez de tempo integral. Esta é uma situação aonde mais é melhor; para Crystal Clear Estou deliberadamente procurando o mínimo que satisfaça a *segurança do projeto* prioridade.**

- **XP exige que a equipas decidir e aderir a convenções de codificação comuns, algo que não é necessário ou mesmo mencionado no Crystal Clear. convenção do XP se torna compreensível quando se considera que os programadores mudar parceiros de programação com frequência, e tem que chegar a um acordo sobre convenções de codificação, a fim de evitar brigas constantes e confusão.**

- **XP fala explicitamente sobre a simplicidade no projeto e refatoração, como parte do episódio desenvolvimento e também a longo prazo. Estes só são recomendados em Crystal Clear, como discutido nas estratégias *caminhando de esqueleto* e *Incremental rearquitetura.* Enquanto eu sou claramente a favor de projetos simples, com refactoring em curso, a questão é, ele deve ser um padrão exigido na metodologia? Minha conclusão é que aprender a simplificar e refatorar design é parte do caminho de crescimento normal de um programador profissional, e não algo para Crystal Clear para tentar legislar em todos os tipos de projetos e tecnologias de programação.**

- **XP não exigem documentação e Crystal Clear faz. Em termos dos dois gols do jogo cooperativo, XP é alto sobre o seu foco no primeiro gol, entregar o software, e em silêncio sobre o segundo objetivo, se preparando para o segundo gol.**

  XP prevê metade do investimento necessário para o futuro: através de rotações programação Pair-, uma nova pessoa que une uma equipas experiente pode chegar até a velocidade no projeto e código muito rapidamente. No entanto, como as pessoas se deslocam para outros projetos ao longo do tempo, há um vazamento de entendimento da equipas, e é bastante provável que os recém-chegados não vai dominar e internalizar o conhecimento tão rápido como ele vaza. Este decaimento é quase inevitável, o que significa que a equipas deve criar outro, exteriorizada informando marcadores, para apoiar o influxo de novas pessoas. Crystal Clear pede a equipas para executar deliberadamente ambos os "gananciosos" e ações "de investimento", equilibrando os caminhos de comunicação de longo prazo com a curto prazo.

Olhando para estes, vemos que aonde XP regula, regula regras mais rigorosas, e aonde é mais solto, é por causa da ausência de uma regra. Isso tem duas consequências:

- **Você está convidado a adotar qualquer das práticas extremas em Crystal Clear. Isto inclui cliente no local, o jogo de planificação, programação em pares, iterações de três semanas, a programação teste anterior, testes de unidade totalmente automatizados, integração contínua, convenções comum codificação rigorosos e desenhos simples com refactoring em curso.**

- **Se você quer fazer full-on XP e ainda atender Crystal Clear, você deve adicionar duas coisas: adicione um compromisso de não apenas uma iteração, mas também *entregar* código regularmente; e adicione lembrando e informando também a documentação, como parte das atividades de investimento envolvidas na configuração para o próximo jogo.**

De um ponto de vista técnicas, XP fornece uma rica biblioteca de técnicas eficazes para o desenvolvedor profissional de dominar.

Crystal Clear e Scrum

Scrum concentra-se em três práticas fundamentais, bem como as propriedades de Fechar Comunicação e Foco:

- ***Time-boxe.* O subconjunto dos requisitos a ser desenvolvido durante cada mês iteração um- está congelado**

  no "backlog iteração" no início da iteração. Isto dá à equipas a paz de espírito (Foco) para trabalhar com eles sem medo deles mudando. Scrum requer uma demo ou equivalente após cada tempo-box, não uma entrega real.

- ***backlog dinâmico.* Para compensar o bloqueio das necessidades durante a iteração, uma lista ("product backlog") é mantido de tudo que falta para fazer. Os patrocinadores estão autorizados a mudar o backlog da forma que quiserem, quando quiserem. A equipas de desenvolvimento e os patrocinadores olhar para o atraso no início de um novo período de tempo-box e selecione o subconjunto a ser congelado por tempo-box. Ou seja, eles *priorizar o backlog* antes de cada iteração de modo que cada iteração tem como alvo o maior trabalho prioritário cada mês 57.**

- ***standup Daily.* A equipas se reúne brevemente (literalmente em pé) cada dia para deixar que cada pessoa anunciar o que ele trabalhou ontem, está pensando em trabalhar em hoje, eo que está segurando-o de volta. Isso vincula a equipas com intercâmbio social e técnica e cria um sistema de alerta precoce para quando sair da pista.**

Estes combinam bem com os três principais práticas de cristal:

---

57 Jeff Sutherland está evoluindo "Scrum contínua." Usando o planificação automatizado e ferramentas de rastreamento, o conteúdo de tempo-box é permitido variam diariamente, não apenas mensalmente (veja http://wiki.scrums.org/index.cgi?ContinuousScrum). Isso requer uma abordagem diferente para a gestão de relacionamento com os utentes e as expectativas do patrocinador. Estou ansioso para seus escritos sobre isso.

- ***entregas frequentes,* não apenas demo'ing o sistema, mas realmente colocá-la nas mãos dos utilizadores, um "user friendly" no mínimo.**
- ***estreita comunicação* para Crystal em geral, e *Comunicação osmótica* para Crystal Clear, facilidade de perguntas e respostas em qualquer momento durante o dia, não apenas durante o standup diariamente, e ouvindo de informações no ambiente de pequena equipas.**

- ***Workshop de reflexão,* ajuste das regras permite a rápida adaptação às condições locais dentro do prazo do projeto.**

Scrum e Crystal se encaixam perfeitamente em conjunto, produzindo o que um engenho chamado *processo não-Process.* O processo Não-Process diz, mais ou menos, começar em qualquer lugar, trabalhar em ciclos curtos com alta de comunicação e feedback reflexivo, e, eventualmente, você vai acabar com o que você precisa (um algoritmo genético, se quiser).

- **Time-boxing com entregas reais é o motor para o processo.**
- **redefinição de prioridades periódica da carteira mantém o produto na pista a longo prazo.**

- **Post-iteração workshops de reflexão manter a equipas e processo no caminho certo no longo prazo.**

- **comunicação osmótica e stand-ups diários manter a equipas no caminho certo no curto prazo.**

Provavelmente, você pode adicionar o processo Não-Process para processo de sua organização, não importa o que é. Seria um algoritmo genético realmente maravilhoso, a não ser que seu projeto provavelmente tem um prazo de 4, 6 ou 10 meses, e você não tem tempo para começar apenas *qualquer lugar* e evoluir para óptima. Você precisa começar *bem perto.*

Cristal ajuda você a começar *bem perto* com-lhe propriedades, princípios, técnicas e exemplos.

Crystal Clear e RUP

O Rational Unified Process (RUP) é muito semelhante à família de cristal de metodologias na estrutura, mas de uma maneira muito diferente do que o XP é.

Ambos RUP e Crystal são "geradores de metodologia." Nem é uma metodologia ou processo por conta própria (sim, eu sei que é chamado o Rational Unified *Processo* mas não é um processo, de qualquer maneira, é um gerador de processo, ou "estrutura de processo' na linguagem RUP). Ambos RUP e Crystal se baseiam na visão de que nenhum processo único pode caber todos os projetos e, portanto, a equipas do projeto tem para personalizar a metodologia para caber sua situação local.

Ambos RUP e Crystal contêm conselhos sobre como custom-tailor a metodologia para a organização e projeto; ambos descrevem técnicas principais e contêm amostras de produtos de trabalho e modelos. Ambos recomendar crescimento incremental do sistema usando o desenvolvimento iterativo, boas práticas de teste, contato próximo ao utilizadores, gestão de riscos e feedback de execução de código.

RUP de *começo* fase é muito semelhante à que se *atividade de afretamento* descrita neste livro. RUP de *casos de desenvolvimento* para customização processo são tão central para RUP como a oficina de metodologia de formação é de cristal (se você não está sintonizando RUP para a sua empresa com os casos de desenvolvimento, então você não está fazendo RUP como se você não fizer metodologia de formação e workshops de reflexão, então você não está fazendo cristal).

Nesse nível de discussão, há muito pouco para distinguir o gerador processo RUP a partir do gerador metodologia de Cristal.

Três elementos de núcleo diferentes. conjuntos RUP *arquitetura, modelagem visual,* e *uso da ferramenta* como elementos de ncleo. Cristal considera arquitetura ser uma questão local, apesar de eu dar a mesma recomendação para construir uma arquitetura cedo, executável. Cristal difere fortemente da RUP sobre o assunto de modelagem visual, usando pela primeira vez a ideia de que a documentação está a ser decidido localmente a matéria e, segundo a ideia de que a documentação deve tentar transmitir a "teoria" do sistema (ver Pergunta 1 em *Questionada),* e modelos visuais são apenas uma pequena parte de transmitir essa teoria. Cristal em geral é uma ferramenta agnóstica, e Crystal Clear até um pouco reticentes quando se trata de ferramentas de modelagem. Os que servem projetos Cristal melhores tendem a ser a gestão configuation, comunicação, programação, equipamento de teste, teste de cobertura de código e ferramentas de desempenho de perfil.

Cristal contém uma prática central que facilmente poderia ser (e deve ser!) Adicionado ao RUP: o *Workshop de reflexão.* Não há nada para parar todas as equipas usando uma instância RUP de simplesmente ficar juntos após cada iteração e refletindo sobre como fazer melhor. Os autores do RUP pode (e deve) basta adicionar uma amostra técnica e trabalho mostrando equipas como fazer alguma forma de Melhoria reflexiva.

Existem outras diferenças, alguns dos quais são essenciais e alguns dos quais são associações mais mentais sobre as práticas da metodologia e sua "sensação" na prática. Essas associações mentais gerar sua própria realidade em organizações que adotam uma metodologia e por isso não devem ser ignorados.

Luz e tolerante. Cada metodologia de cristal se destina, em seus valores centrais, para ser o mais leve, eficiente e não burocrática, e tolerante como a situação do projeto permitirá. Aqueles que não são declarados elementos do RUP. Isto dá o resultado que Andrea Branca descreve: "É possível fazer qualquer processo *Veja* ágil, mas difícil de fazê-lo *sentir* ágil."

Moldar princípios. O cristal contém um conjunto de princípios (descrito em *questionado)* pretende dar orientação sobre como fazer escolhas shaping. Esses princípios são parte do Cristal "pacote." RUP recomenda alfaiataria e lista marcos e modelos para alfaiate, mas não fornece nenhuma teoria sobre como fazer as escolhas shaping (estes podem, naturalmente, ser adicionado, com base na teoria fornecido em Crystal).

Velocidade da metodologia de formação. Cristal pede a metodologia moldar a ser feito é muito pouco tempo, geralmente na ordem de dias, de modo que o custo dessa oficina é reembolsado pelas novas eficiências *dentro do prazo de projeto.* A metodologia de cristal é destinado a evoluir dentro do curso do projeto, o que significa que a metodologia de formação tem de jejuar. Moldar RUP não precisa levar meses, mas muitas vezes o faz, que pode ser

por que tantas empresas a evitar casos de desenvolvimento e adotar o pacote inteiro (que é duplamente ruim, porque eles têm a ineficiência de uma metodologia desafinado com a pena de custo de um excessivamente pesado).

Corta-se contra acumulação. A abordagem em Crystal é começar do que quer que sua equipas vem com na oficina metodologia de formação, ou o que você usou última vez, e construir a partir daí, somando-se a metodologia somente quando experiência revela um problema (tipicamente, durante o *Workshop de reflexão).* Cristal é deliberadamente underspecified, cometer "erros de omissão," se você quiser. A idéia é que ele é mais fácil de detectar quando você está faltando alguma coisa que você precisa do que está a detectar que você tem algo extra que você não precisa. Crystal é deliberadamente "esticar para caber."

RUP, em contraste, é "encolher para caber." RUP contém uma enciclopédia de modelos de produto de trabalho (entre outros conteúdos). Assim como você não iria ler uma enciclopédia da frente para trás, ou preparar todas as receitas em *The Joy of Cooking* em ordem sequencial, você não deve preencher todos os modelos na enciclopédia RUP da frente para trás. Em vez disso, assim como você olhar para um livro de receitas ou uma enciclopédia para o que você precisa hoje, você deve olhar para a enciclopédia RUP e retire os elementos específicos que você precisa neste projeto particular.

A abordagem em RUP, então, é começar a partir da enciclopédia completa e jogar coisas fora durante a atividade de formação. Neste sentido, é overspecified.

Os dois errar de maneiras diferentes para diferentes organizações. Espero grupos a ter problemas com Cristal porque é muito vazio para começar; muitos têm problemas RUP porque é demasiado cheio para começar. Eu não vejo nenhuma fuga de um dilema desse tipo.

A modelagem visual. Tal como descrito acima, modelagem visual é um elemento de núcleo de RUP, aonde se trata de um elemento electiva em cristal.

Ferramentas. RUP é projetado para ser apoiada por ferramentas, aonde Cristal está deliberadamente agnóstica em ferramentas, e até mesmo um pouco contra o investimento em ferramentas de modelagem visual. Minha recomendação, desde 1992, tem sido a de comprar qualquer ferramenta permite desenhar formas conectadas a mais rápida; que inclui lápis e papel, quadros brancos e câmeras, Powerpoint, Visio e ferramentas de CAD-CAM (alguns dos quais, segundo me disseram, tem usabilidade excelente).

desenvolvimento simultâneo. Cristal tem desenvolvimento simultâneo como uma recomendação do núcleo, com orientação para a incorporação de requisitos just-in-time quando ele se adapte ao seu projeto. RUP contém recomendações para o crescimento incremental do sistema, mas nenhuma orientação sobre como incorporar requisitos just-in-time e desenvolvimento simultâneo.

apoio corporativo. Não há como escapar a diferença de que o RUP é apoiado por uma equipas de design dedicado, juntamente com marketing, desenvolvimento de ferramentas, treinamento e consultoria braços. Esta não é uma diferença na definição de metodologia, mas ele cria uma diferença muito grande em adoção. Você não pode comprar uma cópia do Crystal; você só pode comprar uma cópia de dois livros (*Desenvolvimento Ágil de Software* e este) e invadir meu site

(Alistair.Cockburn.us). Depois que você e sua equipas tem que se reunir e *pensar, discutir* e *refletir. (* OK, para um número muito limitado de equipas, um especialista em metodologia de cristal pode ser contratado para fazer algum treinamento). Isso é muito diferente de colocar uma ordem para livros, ferramentas, cursos e consultoria para mostrar-se à porta da sua organização.

você pode fazer Crystal Clear dentro RUP? Será que Crystal Clear ser uma implementação válida do RUP, assumindo, claro, uma equipas colocated de três a oito pessoas?

Lembre-se que apenas os primeiros três propriedades da Crystal Clear são obrigatórios. Tudo o resto é consultivo ou, pelo menos em discussão. Estas três propriedades não são conceptualmente difícil; a questão é se uma equipas fazendo RUP vai realmente pôr-se a entregar a um utilizador real a cada poucos meses, se a equipas vai se deslocar de seus gabinetes para uma área comum para obter Osmótica Comunicação, e se eles vão sentar, discutir em um *workshop de reflexão* e alcançar Melhoria reflexiva. Se eles podem fazer essas três coisas, então, provavelmente, o resto do que eles fazem vai cair dentro Crystal Clear.

Não é óbvio para mim que Crystal Clear, como está escrito, é uma implementação válida do RUP. Pode não haver modelagem bastante visual ou arquitetura feito ou gestão de risco explícito para passar no teste RUP. Se ele faz ou não faz, provavelmente, depende de como seus casos de desenvolvimento de sair.

Pergunta 7. E sobre o CMM (I) 58?

Crystal Clear não foi concebido com a intenção de requisitos do CMM (I) de reuniões, e é difícil ver como ele iria subir essa escada. Crystal Clear se refere a muitas das atividades chamou na CMM (I), embora talvez não na forma como o CMM (I) assessor padrão é usado para. No entanto, Crystal Clear tem um foco do projeto e da CMM (I) tem um foco organizacional.

Por outro lado, muitas organizações já certificadas na CMM (I) nível três ou acima deve ser capaz de, quer introduzir Crystal Clear ou pedir emprestado a partir dele para aumentar a eficiência. Vejamos que mais perto, imaginando uma organização já fazendo o desenvolvimento de software no nível 3 do CMM (I) e querendo usar Crystal Clear.

- **Eles têm de encontrar um utilizador "amigável" disposto a deixá-los implantar o sistema a cada poucos meses. Este deve violar nenhuma das suas regras de processo. No entanto, os grupos que encontrei que priorizam CMM certificação (I) colocaram seus desenvolvedores tão longe dos utilizadores e sobrecarregados seu processo de tanto que é improvável que eles podem entregar corrida, testado software a cada trimestre.**

- **Eles têm que colocate a equipas para obter Comunicação osmótica. Isso deve quebrar nenhuma regra de processo. No entanto, muitas organizações CMM (I) estão comprometidos com equipas distribuídas.**

- **Eles têm de adoptar Melhoria reflexiva. Ele deve violar nenhuma regra de processo que as pessoas se reúnem e procuram maneiras de trabalhar melhor. O problema pode causar é que a equipas é provável que sugerem mudanças no processo. Muitas organizações que têm ido para a dificuldade de obter CMM (I) certificado não deixou espaço para o processo de mudança. A mudança para um processo evolutivo, embora difícil, será um saudável. Em todos os três casos, a mudança necessária não está nas regras de processo, mas na mentalidade (o mais difícil dos dois para mudar).**

Frequentando Foco, Segurança pessoal, e Fácil acesso a utilizadores experientes não deve comprometer qualquer parte de seu processo declarada, mas apenas ajudar as pessoas a trabalhar melhor. Da mesma forma, o Ambiente técnica com Automated Testing, Gestão de Configuração e Integração Frequent Pode levar algum trabalho, mas não deve interferir de alguma forma com o processo de certificação.

Os produtos de trabalho de Crystal Clear é consultivo para começar, e a organização certificada no nível 3 é susceptível de ter todo o ones nome que eu (e outros além). A organização pode adotar algumas das ideias apresentadas na seção dos produtos do trabalho para simplificar ou melhorar o seu conjunto de produtos de trabalho (substituindo os gráficos da programação de Gantt com valor ganho ou *Queime* gráficos seria um bom começo).

---

58 CMMI é mais amplo em escopo e mais flexível do que o "CMM para Software." Na verdade, o CMM para Software não é mais suportado oficialmente. Para os fins desta discussão, porém, não há diferença significativa entre o CMM e CMMI.

Existem várias outras diferenças entre Cristal e do CMM (I). A mais óbvia delas é que o CMM (I) está hospedado e apoiado por uma organização de tamanho considerável, que atua como um recurso para a formação e exame, enquanto Cristal é sustentada por três alguns livros e um punhado de consultores especializados em todo o mundo. Mesmo que com boa sorte Cristal vai ter mais apoio, é improvável que nunca ter apoio organizacional ao mesmo nível que o Instituto de Engenharia de Software (se isso acontecesse, ele teria que ser chamado de Gamesmanship Software Institute!).

O verdadeiro núcleo das diferenças vem dos valores subjacentes e suposições sobre desenvolvimento de software. O CMM (I) é fundamentalmente sobre a consistência, a previsibilidade e reprodutibilidade. Cristal é fundamentalmente sobre a eficiência, a habitabilidade, e conjuntos de regras otimizados localmente.

literatura de engenharia de processo categoriza processos como seja *definido / teórica (* não o CMM (I) significado de "definido" Apresso-me a acrescentar!) ou *empírico.* UMA *definido / teórica* processo é aquele que é suficientemente bem compreender a ser automatizado. A *empírico* é preciso exame e intervenção humana. Um livro no campo tem que dizer isto:

> **"É típico para adotar a abordagem de modelagem definida (teórica), quando os mecanismos subjacentes pelos quais um processo opera são razoavelmente bem compreendidos. Quando o processo é muito complicado para a abordagem definida, a abordagem empírica é a escolha apropriada." (Ogunnaike 1992)**

Para retirar um *definido / teórica* processo, é preciso conhecer e ser capaz de medir adequadamente todos os parâmetros relevantes, mas mais significativamente, o sistema deve ser não caótico. Em um sistema caótico, até mesmo a menor perturbação em um parâmetro chave envia o sistema fora em uma direção diferente; nenhuma quantidade de medição e de controlo é o suficiente. O (I) em escada CMM é construído sobre a suposição de que estas condições forem satisfeitas.

Pessoalmente, duvido que o primeiro ou o último pressuposto é cumprida no desenvolvimento de software. Primeiro, eu não acho que nós sabemos os parâmetros relevantes para medir. Em segundo lugar, eu acho que é bastante provável que qualquer atividade criativa baseada em equipas é caótica, o que significa que uma pequena perturbação pode causar um efeito arbitrariamente grande. Aqui está um pequeno exemplo:

> **Em uma reunião sobre uma nova iniciativa, as pessoas na sala estavam tensos, mas aparentemente bem-comportado. Quando tomámos uma curta pausa, a mulher que estava a liderar a nova iniciativa se transformou em sua demissão! Parece que um dos (para mim) farpas levemente formuladas oferecidas pelo futuro CFO era simplesmente "um muito" por ela, e ela deixou no local.**

Pense na última vez que você fez o que seu pensamento foi um comentário inócuo e seu ouvinte tem surpreendentemente chateado. Pense em quando uma pessoa chave faltou a uma reunião. Tanto de quem pode cascata em um arbitrariamente grande consequência para o projeto.

A necessidade de ter pessoas detectando coisas acontecendo de errado e intervir de forma inesperada foi levado para casa para mim em um e-mail a partir de Glen Alleman, informando sobre uma organização (CH2M HILL Informação e Comunicação Solutions), que fornece aplicações de manejo de material nuclear, ERP, segurança cibernética e outro

serviços a vários milhares de utilizadores sob as diretrizes da Instalações Nucleares Safety Board Defesa (que controla milhares de edifícios contaminados com produtos químicos e dos resíduos radioactivos). Glen escreveu:

> **Acabamos de vir de uma "segurança se retirar," (há sempre algum problema de segurança com o nosso negócio, incluindo a parte de TI). Nas aulas processo de uma aprendeu das perguntas é: "Se você seguiu o procedimento corretamente que o problema não teria ocorrido?" A resposta dos caras de segurança é essencialmente - há processos empíricos que devem estar presentes.**

Esta única pergunta permite que a organização para detectar novos parâmetros, anteriormente insuspeitas de importância. Em qualquer nível de realização, uma equipas deve ter a humildade e ceticismo saudável para fazer esta pergunta e siga aonde ele leva.

Pelas razões acima expostas, acho que é improvável que o problema nunca vai surgir seriamente de uma CMM (I) organização Nível-3 fazendo Crystal Clear, embora um (I) grupo -Certified CMM pode emprestar ideias de Crystal Clear.

* * *

Este é o tempo para explicar o que quis dizer na seção anterior por Crystal transformando o (I) pirâmide CMM cabeça para baixo.

Cristal é construído no conceito de que *cada* metodologia e organização precisa ser *auto-evolução.* O que os lugares CMM (I) no nível superior, otimizando, eu tenho como um requisito inicial. A cada usos de projeto de cristal Melhoria reflexivo, guaranting que as regras vão mudar ao longo do tempo. Se o Crystal estavam a ter uma atividade de certificação, em seguida, mesmo em "Nível de cristal 2", a equipas teria que mostrar que eles *mudado* seu processo dentro e através de projetos!

Há uma inversão menos distinta da pirâmide, que tem a ver com a precisão de medição. A fim de entrar em controle estatístico de processos, o CMM (I) exige medidas numéricas para acontecer no nível 4. O conceito que eu encontrar ausente nesta abordagem à medição é a ideia de *tornando as medições mais precisas em níveis mais elevados,* começando com "medidas fuzzy" que são nada mais do que adjetivos ou expressões de sentimentos.

Quando uma pessoa diz em um *workshop de reflexão* ou reunião de status "que correu bem", "Eu não estou confortável", "eu não gosto", "Eu não estou completamente certo", ou "houve uma wobbliness no início", ela está oferecendo a resposta integrada de um profissional que tem sido em muitas situações em suas vidas. É um rolamento de uma grande quantidade de informações recolhidas a partir de uma série de fontes, em uma única expressão. É um "número fuzzy" no sentido matemático (Lógica Fuzzy ref ???), ocupando uma faixa de valores com uma função de distribuição de probabilidade.

Estes são os grandes valores de medição para muitas situações, particularmente quando os parâmetros importantes são numerosos ou desconhecidos, como com um projeto de desenvolvimento de software.

A pirâmide, que seria interessante para mim começaria com medições pessoais, distorcido do projeto nos níveis mais baixos. A maioria dos projetos se beneficiariam muito de gravação e analisá-los. Como a organização progride, eles podem aprender quais são mais significativos para rastrear e encontrar maneiras de apontar o valor de medição, tornando-a mais *preciso,* eventualmente, acabar com os números (que eu considero afiada, coisas pontiagudas quando aplicado a coisas como humor projeto).

Pergunta 8. E sobre UML e Arquitetura?

A Unified Modeling Language (UML) é um padrão de notação desenho, não uma metodologia. Em termos do quadro de metodologia padrão (papéis, técnicas, padrões, metas, etc.), é um dos padrões para um ou dois dos papéis. É um componente de uma metodologia, mas não é provável que seja um grande fator no resultado do projeto.

O padrão UML não faz prescrições ou até mesmo recomendações sobre quando, aonde e quanto você desenhar. Com relação ao padrão, você pode tirar todo o projeto antes de começar a programar, ou você pode "pensar um pouco, desenhar um pouco, código um pouco," uma estratégia de longo recomendado por especialistas em tecnologia objeto e apoiado pelo livro de Scott Ambler *Agile Modeling (* Ambler 2002).

Discussão de modelos de desenho rapidamente se vê envolvido na discussão da arquitetura e quanto projeto de fazer quando. Você não vai se surpreender ao saber que o meu ponto de vista é que diferentes pessoas têm diferentes preferências e gosto de pensar sobre projetos para diferentes quantidades de tempo antes de começar a programa.

Dito isto, também é minha opinião de que um projeto é uma "teoria", que precisa desesperadamente de um "experimento" matching para validá-lo, ou, mais importante ainda, revelar suas fraquezas. Normalmente, o projeto não sobreviver seu primeiro encontro com código real. Isto significa que quanto mais cedo o projeto é testado na implementação, quanto mais cedo o designer aprende aonde e como ele precisa ser melhorado.

Algumas pessoas têm um horizonte temporal de pensamento muito curto antes de se sentirem desesperadamente a necessidade de uma experiência: na ordem de cinco ou dez minutos. Outros têm um horizonte de médio prazo, na ordem de várias horas ou um dia ou dois, durante o qual eles mentalmente explorar todas as armadilhas, fraqueza e solicitações de mudanças potenciais que possam ter de se adaptar. Outros têm bastante longo pensar horizontes de tempo, da ordem de uma ou duas semanas, durante o qual eles fazem o mesmo. Minha experiência é que quando os designers gastar mais do que algumas semanas de trabalho para fora um design (software), eles costumam ter passado do ponto trade-off ideal, aonde teria sido útil para criar uma experiência com código real para obter algum feedback verdadeiro 59.

Tomando todos aqueles juntos, eu sinto que a maioria dos arquitetos levar muito tempo antes de fazer sua primeira experiência, mas não é necessário adoptar o ponto de vista que o design pode ser substituído apenas por refatoração e atenção para "uma única vez" regra XP do.

59 Eu ouvi uma história de um programador atribuído a programar um elemento kernel do sistema operacional, que olhou para seu projeto por meses, estudando a sua simetria e estética, até que ele estava totalmente convencido há simetria estava faltando e não poderia ser mais simples, pelo que ponto ele digitou (e funcionou). Tal situação é extremamente rara.

Os sujeitos da UML, desenhos, ferramentas, projeto arquitetônico, e pensando horizontes de tempo tendem a ficar confuso todos juntos. Se você pode separá-los, ele libera-lo para trabalhar em novas combinações.

É, por exemplo, *não* o caso que todos os desenhos são desenhos UML (ver os produtos de trabalho para alguns exemplos), nem deve arquiteturas ser descritos em UML, ou mesmo em desenhos, nem mosto ferramentas de software extravagantes ser usado para capturar qualquer desenhos ou UML (imagine UML desenhado à mão , consulte os produtos de trabalho para exemplos), nem gastar tempo pensando significa sentado em uma ferramenta. Finalmente, e em especial, nem é o caso que sentado na frente de uma ferramenta de desenho UML significa que qualquer pensamento arquitetônico está acontecendo.

A abordagem típica em um projeto de Crystal Clear é usar o *caminhando de esqueleto* e *rearquitetura incremental* estratégias. Uma arquitetura precoce é esboçado, programado e testado na primeira iteração, e completou, ampliou e evoluiu ao longo iterações subseqüentes. UML pode ou não ser usado para pensar por ele e documentar o resultado.

Pergunta 9. Por que visam apenas para a zona de segurança? não podemos fazer melhor?

Sim, você pode fazer melhor, e se você estiver por cima dela, encorajo-vos a fazê-lo. Por exemplo, você pode aplicar todas as práticas XP, com o desenvolvimento orientado a testes, em tempo integral a programação em pares em rotações, automatizado testes de aceitação, bem como testes de unidade, utilizadores experientes e especialistas de domínio no local em tempo integral, contínua e compilações automatizadas , encurtar o ciclo de iteração para uma ou duas semanas, e entregando mensal.

Você pode, e deve, se possível, instituir um grupo semanal de discussão de uma hora. As pessoas que fizeram este relatório de volta no aumento da confiança, capacidade e a capacidade das pessoas para discutir seus temas com o outro. Trabalhe seu caminho através de vários livros, incluindo *O programador pragmática (* Caça 2001), os livros sobre teste-driven design (Beck 2003 Astels 2004), refatoração (Fowler 2002) e qualquer número de livros padrão design. Nestas sessões, mostrar e discutir trechos de código do projeto, aprender a analisar e comparar as técnicas de programação de design. Discutir formas de código de teste. Use as sessões de experimentar novas linguagens e ferramentas.

Você pode e deve pagar extra atenção especial à Segurança pessoal propriedade, e construir a base para Confiar em entre as pessoas - técnicos, bem como a confiança pessoal. Você pode e deve alocar tempo e dinheiro para a formação profissional dos membros da equipas, para que eles manter-se atualizado com o nosso campo de movimento rápido. Por exemplo, tenho totalmente revisto minhas técnicas de programação a cada dez anos - Estou no meu quarto ciclo de reaprendizagem, e que é apenas para a programação. Eu nem sequer tocou usabilidade ou design de interface de utilizador, no entanto, e estou terrivelmente fora da data em sistemas de gestão de configuração. Aonde você está? Se você não alterou significativamente os seus hábitos de design, programação, design de interface do utilizador e teste nos últimos cinco ou sete anos, você está quase certamente muito desatualizado.

A lista de como fazer *Melhor* do que eu têm exigido no Crystal Clear é longa. No entanto, cada uma dessas coisas acrescenta dificuldade na adoção e reduz o número de pessoas que podem gerenciar a soma total.

A família de cristal de metodologias não é direcionado para ser "ideal" para qualquer uma das dimensões de possíveis prioridades do projeto. Não otimizar a produtividade, ou para a rastreabilidade, a responsabilidade legal, a liberdade de defeitos, descontraído backness, equipas distribuídas, uso máximo de equipas júnior, ou qualquer uma das outras prioridades que você venha com.

Cristal tem como prioridades: segurança do projeto, eficiência e habitabilidade. A prioridade eficiência diz, jogar o jogo económico-cooperativo bem, permitindo uma boa velocidade de entrega neste jogo enquanto ainda prestando atenção para o próximo jogo. A prioridade habitabilidade diz, ter em conta os pontos fortes e fracos dos seres humanos que trabalham em conjunto naturais, e para tornar o resultado razoavelmente agradável para se viver. De segurança Projeto significa que o software vem para fora da porta em um prazo justo, com o pessoal do projeto intacto e disposto a passar pelo mesmo processo novamente.

O conflito de eficiência e de habitabilidade prioridades (como tantos problemas fazer isso temos de participar no desenvolvimento de software). Um pode ser mais eficiente por ser menos tolerante. Um pode ser mais tolerante, exigindo menos eficiência.

Felizmente, este livro fornece rigor suficiente para a sua equipas para entrar em zona de segurança do projeto, e latitude suficiente para que você pode adicionar o que prioridade seguinte você precisa em seus projetos.

Pergunta 10. E sobre equipas distribuídas?

Esta é uma questão muito mais interessante agora (2004) do que era quando eu formulada pela primeira vez Crystal Clear.

Crystal Clear é baseada em ter Comunicação osmótica. Isso me leva seis segundos para empurrar a minha cadeira para longe, caminhe fora do escritório, a um escritório um para baixo e outro lado do corredor, e cumpri a minha cabeça em perguntar à pessoa uma pergunta.

Em seis segundos, eu ainda pode descobrir a pessoa não está lá e voltar para minha estação de trabalho e continuar trabalhando sem perder minha linha de pensamento. Ou pedir a pergunta, cara a cara. O ponto é para limitar a quantidade de tempo e energia utilizada para obter a questão colocada, e têm curta rica conversa, sobre isso.

Richard Herring informou sobre um experimento no qual a equipas, sentado em andares diferentes do mesmo edifício, microfones utilizados e web câmeras para simular "presença e consciência" (Herring 1999). Cada pessoa colocar a foto na webcam de cada outra pessoa como pequenas imagens em suas estações de trabalho. Com estas pequenas imagens, eles poderiam cada um ver se a outra pessoa estava ocupado com digitação cabeças-down, em discussão, ou ausente. Com os microfones e alto-falantes, eles poderiam fazer e responder perguntas imediatamente. Usando a tecnologia de bate-papo, eles poderiam fazer e responder perguntas em silêncio e de forma assíncrona (reduzindo o fator de perturbação ainda mais). Eles ainda copiado código de e para o seu ambiente de desenvolvimento, para que eles pudessem trocar código real e não apenas sugestões sobre o assunto.

Richard Herring satisfez as minhas preocupações sobre Osmótica Comunicação, e é claro que seu grupo tomou conta de sua vida social Segurança pessoal e problemas de confiança também. Esta foi uma maneira muito criativa para lidar com um dos "pontos doces" (ver a primeira seção *questionado* para uma discussão sobre os pontos doces e simulando-los). Se você tem uma única equipas levemente distribuídos, tente ideias de arenque.

Victoria Einarsson, na Suécia, descreveu o uso de algo essencialmente como Crystal Clear, exceto que os desenvolvedores trabalharam fora de suas casas. Victoria descrito passar horas a cada dia no telefone, fazer e responder perguntas e manter as conexões sociais no lugar. Várias coisas ficaram claras durante a nossa conversa. Primeiro, eles eram realmente eficaz como uma equipas. Segundo, eles imitaram Crystal Clear em quase todos os aspectos. Em terceiro lugar, não era realmente Crystal Clear porque eles não têm

Comunicação osmótica. Em quarto lugar, o seu sucesso foi fortemente dependente de tremendas habilidades sócio-técnicos de Victoria do líder da equipas. Em outras palavras, se eles queriam criar uma segunda equipas do projeto, eles teriam que ter muito cuidado em entrevistar o novo líder do grupo por ter excelentes habilidades sociais e técnicas. Eles foram bem sucedidos, mas não é uma configuração de projeto que eu seria facilmente recomendar, porque era tão dependente de habilidades de Victoria.

O sucesso desta equipa levanta uma questão relevante. Não é o caso que você tem que fazer tudo em Crystal Clear para ser bem sucedido. Um número de equipas que visito são bem sucedidos

enquanto soltando elementos de Crystal Clear. Eles operam fora da zona de segurança geral que Crystal Clear descreve, e fazer-se de que por ter algumas pessoas de destaque em posições-chave na equipas. Ter essas pessoas também é uma maneira de ganhar o jogo, mas não o tema deste livro em particular.

Equipas distribuídas em mais do que apenas uma cidade são mais dependentes da tecnologia de comunicação e indivíduos excepcionais. Existem maneiras para eles para ter sucesso, mas essas formas não são Crystal Clear.

Pergunta 11. E quanto a equipas maiores?

Eu tive o prazer de visitar e entrevistar Jan Siric, depois de Nordea na Dinamarca, sobre seu projeto de 16 pessoas, que combinava com todas as características de Crystal Clear exceto para o tamanho da equipas.

> **Os doze desenvolvedores, a programação tanto mainframe e software estação de trabalho, sentou-se em três longas mesas, dois de cada lado de cada mesa comprida. Cada mesa tinha uma estação de trabalho em cada extremidade. A secção central de cada mesa era um quadro deitado de lado. Os monitores foram montados em trilhos para que eles pudessem ser deslizar para dentro da seção central. Jan e outros três da equipas líder / gerente pessoas se sentavam em mesas ao lado. A mesa de conferência longo estabelecem novas da sala. O quarto era arejado, espaçoso e com as plantas. Eu imediatamente notado a presença de carpetes no chão e telha acústica no teto para atenuar o ruído. Como resultado, o ruído não foi um problema.**

Com esse arranjo, Jan foi capaz de chegar a seis segundos de tempo, mesmo com 16 pessoas, e as pessoas poderiam trabalhar sozinho, em pares, em grupos ou como um grupo como eles precisavam.

O que mais me impressionou, no entanto, foi a capacidade de Janeiro para detectar rachaduras na comunicação. Ao ouvir um comentário estranho no refeitório um dia, ele entrevistou os programadores e descobriu que um programador de mainframe que tinha apenas recentemente se juntou a sua equipas, não tinha totalmente ajustado às suas conversas abertas com perguntas e interações frequentes. Jan reorganizados os assentos, trocou os chefes de equipas para que aquele com as melhores habilidades sociais e de comunicação foi responsável por essa subequipas particular, e, em seguida, prestou especial atenção à forma como as interações prosseguiu.

As ações de Jan perfeitamente ilustrar o conceito de julgar o estado do projeto, em boa medida pela qualidade das comunicações de Crystal Clear.

No entanto, eu não recomendaria Crystal Clear para equipas de 16 pessoas como uma regra geral. Jan foi capaz de retirá-lo porque ele foi excelente para as questões sociais / comunicação envolvidos, e capaz de obter tanto Comunicação osmótica e Segurança pessoal no lugar com essa equipas particular.

Em circunstâncias comuns, Comunicação osmótica é muito difícil de alcançar com mais de cerca de oito pessoas. grupos XP pode alcançá-lo com 12 pessoas, utilizando seis estações de trabalho e duas pessoas por estações de trabalho. Depois disso, torna-se bastante difícil.

Crystal Clear não se destina à escala. É uma forma eficiente de trabalhar quando você pode conseguir colocation e Comunicação osmótica. Se não for possível atingir esse ponto doce, então você tem que trabalhar de forma diferente.

Pergunta 12. E quanto a projetos de preço fixo e de escopo fixo?

Há uma tendência a pensar que processos ágeis só se aplicam a projetos exploratórios. Isto simplesmente não é verdade. Todas as minhas primeiras experiências foram em projetos de escopo fixo. Comecei a usar estas técnicas, simplesmente porque não havia outra maneira de cumprir o prazo do projeto do que ser super-eficiente, uma vez que estas ideias permitir que um ser. Você deve ser capaz de usá-los, ou adaptações destas técnicas em praticamente qualquer pequeno projeto (muitas das estratégias e técnicas podem ser utilizados ou adaptados para quase todo o projeto).

Vamos ver como *Planificação Blitz* ajusta para projetos em tempo fixo e de escopo fixo.

O projeto em tempo fixo. **Você coloca as cartas na mesa e percorrer as tarefas e timing. A resposta sai maior do que o tempo permitido na programação em tempo fixo. Uma das duas coisas podem acontecer:**

- **Você e seu *Patrocinador executivo* ser criativo com os cartões e encontrar uma maneira de cumprir o prazo.**

- **Apesar de toda a sua criatividade, você não pode encontrar uma maneira de atender o cronograma. A primeira situação mostra uma boa aplicação do *Planificação Blitz* técnica. É possível que a técnica lhe permitiu encontrar uma estratégia que você poderia ter esquecido.**

A segunda situação não mostra uma falha no *Blitz Planificação,* Crystal Clear ou técnicas ágeis. Isso mostra que alguém fez uma oferta errado no projeto. Se você pode convencer os patrocinadores para alterar o lance depende de fatores locais à sua situação. O resultado pode ser de fato que você tem que viver com o cronograma determinado e simplesmente trabalhar horas extras. A diferença é que todos na sala conhece a situação inicial.

> **Em um projeto que foi e pegou o patrocinador do lado do cliente e mostrou-lhe os cartões. Depois de trabalhar com as cartas, ele concordou que o projeto simplesmente não poderia ser feito no tempo determinado, e mudou os termos do projeto. Eu só posso esperar que vocês são tão afortunados.**

O projeto de escopo fixo. **Está a iniciar o segundo, terceiro ou quarto iteração. Você coloca as cartas na mesa, e. . . *você não alterar o escopo do projeto!* Não há nada no desenvolvimento ágil que diz que você *tem que* alterar os requisitos de cada iteração. Crystal Clear e XP dizer que *E se* você está em uma situação aonde você precisa alterar o escopo, este é o momento e a maneira em que para fazê-lo. Para a situação escopo fixo, basta deixar o escopo sozinho**

Pergunta 13. Como posso classificar como "ágil" ou como "Crystal" nós somos?

Os defensores do desenvolvimento ágil teme medir como "ágil" uma equipas de projeto é. No entanto, o gerente de uma carteira de projetos vai querer acompanhar como diferentes equipas adotar elementos da Crystal Clear ou algum outro método ágil. Aqui é como nós respondeu à pergunta de uma só empresa. (Figura 7-5 mostra os resultados de uma série de projetos em diferentes empresas, em diferentes níveis de adopção).

- **Caracterizamos cada projeto por quantas pessoas estavam sendo coordenados e o comprimento iteração. duração da iteração é secundária a frequência de entrega e taxa de visões de utilizadores no meu ponto de vista, de modo que incluía apenas iteração comprimento para caracterizar os projetos, não para avaliá-los.**

- **Listamos as propriedades, começando com os sete propriedades do Capítulo 2. Nós desempacotado-los para expor seus componentes. Assim, visões de utilizadores, testes automatizados, gestão de configuração, frequência integração e colaboração além das fronteiras tornaram itens de rastreamento separados.**

- **Identificamos que estratégias e técnicas pode ser interessante tentar. Na Figura 7-5, mostro uma dúzia vale a pena considerar. Enchemos esta tabela, marcando *com que frequência* as entregas, as oficinas de reflexão, as visões e integrações tinha acontecido. comunicação osmótica tem um Sim se as pessoas estivessem na mesma sala, caso contrário, eu escrevi abaixo propagação da equipas. Segurança pessoal tem '0', '1/2' ou Sim, dependendo de uma avaliação subjetiva. (Se você não tem segurança pessoal em seus projetos, você pode enviar um '0'?).**

Para foco, nós colocamos "prioridades" se isso foi conseguido, "tempo" se isso foi conseguido, ou Sim se ambos foram atingidos. Para o teste automatizado, que considerado unidade e os testes de aceitação separadamente (nenhum tinha automaticamente ensaios de aceitação).

Figura 7-5 reflete a idéia de que é desejável para alcançar todas as propriedades, mas a lista de estratégias e técnicas indica apenas que a equipas pode tentar em seguida. Eu não tenho certeza se é possível ou desejável para chegar a uma pontuação global, mas eu acho que esta tabela pode ajudar um gerente de equipas ou departamento orientar um ou mais projetos. Eu, é claro, postá-lo como um radiador de informações.

Projeto	*EB*	*TÃO*	*EV*	*GS*	*º*	*Ideal*
Pessoas #	25	25	16	6	2	<30
comprimento iteração 2 semanas		1 mês 3 semanas		1 mês 1 mês <2 meses		
Entrega Frequent 2 semanas		1 ano!	1 ano!	1 mês 4 meses! <3 meses		
visões de utilizadores 2 semanas		1 semana	1 ano!	1 Mês 1 Mês <1 mês		
Workshops reflexão	2 semanas	1 mês 3 semanas		0	Um mês <1 mês	
Comunicação osmótica	1 piso	1 piso	sim	sim	sim	sim
Segurança pessoal	1/2	sim	1/2	sim	sim	sim
Focus (prioridades, tempo)	prioridades	sim	sim	prioridades	sim	sim
Fácil acesso a utilizadores experientes	0	1 dia / semana	0	0	sim	sim
Gestão de configurações	sim	sim	sim	sim	sim	sim
Teste automatizado	0	0	unidade	0	unidade	sim
Integração Frequent 3 semanas		3 semanas	diariamente	1 / Semana	1 dia	contínuo
Colaboração através das fronteiras dos	sim	sim	0	0	1/2	sim
Exploratórios 360 °	0	sim	0	0	sim	sim
Victory início	sim	sim	sim	0	sim	sim
caminhando de esqueleto	sim	sim	sim	0	sim	sim
rearquitetura incremental	sim	sim	sim	0	sim	sim
Radiadores de informação	sim	sim	sim	sim	sim	sim
A programação em pares	0	0	sim	0	0	talvez
Side-by-side de programação	0	0	0	0	0	talvez
desenvolvimento Test-primeiro	0	0	sim	0	0	talvez
Planificação Blitz	sim	sim	sim	0	sim	sim
Diário stand-up encontro	sim	sim	sim	sim	0	sim
Interação essencial desenhar	0	0	0	0	0	sim
queime gráficos	0	sim	0	0	sim	sim

Figura 7-5. Uma maneira de controlar o uso das ideias ágil dos projetos.

Pergunta 14. Como eu começo?

Fazer essas quatro coisas imediatamente:

1. Comente as restrições para Crystal Clear (o *09 de junho* enviar e-mail a partir de cristal para Alistair).

Se você não pode atender a essas restrições, vá em frente e continuar com os próximos passos, mas tenha em mente que você vai precisar de regras ou práticas adicionais para compensar os desaparecidos características do projeto.

2. Segure uma prática *workshop de reflexão* com alguns colegas amigáveis (este é um*Processo Miniatura* para o *Workshop de reflexão).*

3. Continue com o *Entrevistas projeto* e equipas adequada *workshop de reflexão* Como descrito na secção de técnicas. Naquilo *Workshop de reflexão,* percorrer descrição deste livro de ciclos de processo, e os produtos de trabalho. Rever a estrutura da equipas e Convenções trabalhar produto, em particular.

4. Identificar quem irá servir como o "user friendly" ao seu grupo, para rever o seu sistema à medida que cresce, antes que se torne algo que o seu pode implantar a todos os utilizadores. Após essas etapas, você deve ter discutido o que significa estar fazendo Crystal Clear, aonde os pontos fortes de sua equipas mentir, e aonde a equipa e os pontos fracos do projeto mentir. Você vai naturalmente têm discutido Testes automatizados, Gestão de Configuração, Integração Frequent e as formas de alcançar Foco. Vai levar algum tempo para Segurança pessoal a ser estabelecida.

5. Executar uma iteração curta. Torná-lo artificialmente suma, uma ou duas semanas, de modo que você pode realizar o seu primeiro *workshop de reflexão* imediatamente e rever seus hábitos de trabalho já (este curto iteração fornece-lhe com um *Miniature processo* em si Crystal Clear). Ele também irá cimentar a *workshop de reflexão* como uma técnica de grupo, fornecer uma saída para perguntas e ideias, e aumento Segurança pessoal (as pessoas vão ver como suas ações são interpretadas).

Neste ponto, você está em andamento. Utilize este e outros livros como enciclopédias de ideias para tentar durante sucessivas iterações. Finalmente,

6. Aguarde *Workshops reflexão* a cada duas semanas para um mês ou dois, e então decidir sobre o intervalo a ser usado entre a próxima *Workshops reflexão.*

Lá. Você está fazendo Crystal Clear.

Capítulo 8 Testado (Um estudo de caso)

Stephen Sykes de Thales Pesquisa e Tecnologia no Reino Unido experimentou com uma versão inicial deste livro e tentei sair. Aqui está o seu relatório sobre a experiência, juntamente com as recomendações do auditor ISO 9001. Muito obrigado a ambos Stephen e Thales.

É um deleite raro para ser capaz de incluir neste livro um relatório campo de alguém que tentou Crystal Clear. É uma ainda maior deleite para ler o relatório do auditor ISO 9001 que analisou o projeto.

Thales Pesquisa e Tecnologia do Reino Unido é uma organização certificada ISO 9001. Querendo saber mais sobre desenvolvimento ágil como parte de sua atividade de melhoria de processos corporativos, eles decidiram tentar Crystal Clear em um programa para calibrar a posição da câmera para um sistema de reconhecimento de cena natural. Stephen Sykes, o designer-chefe inicial para o projeto, baixado e trabalhou um (2002) primeira versão deste livro para esse experimento. A parte final do experimento foi de ter um dos seus ISO 9001 auditores analisar o projeto para o que deve ser feito de forma diferente para quando ele não seria um experimento.

A prioridade para a equipas do projeto foi a de completar o protótipo até o final de janeiro de 2004. O ideal seria ter um produto liberado qualidade. Na verdade, eles acabaram com um demonstrador de boa qualidade que pode ser transformado em um produto razoavelmente curto aviso prévio. O patrocinador terminado o projeto naquele momento, elegendo a cair a partir do escopo do projeto o julgamento de um segundo algoritmo de calibração, e certas atividades de documentação do projeto.

O auditor ISO notou a diferença entre o escopo do projeto final e o original com a descrição:..." .O projeto foi infelizmente interrompido precocemente No entanto, devido à natureza do método de CC, a rescisão antecipada não proibir um produto de software útil ".

Em termos de atenção de Crystal Clear às prioridades do patrocinador, isso conta como um resultado do projeto satisfatória ("... No último workshop de reflexão, o patrocinador também disse que as vantagens superavam em muito as desvantagens. Ele sugeriu que tente isso de novo, em um projeto maior, desde que o ajuste com os nossos procedimentos existentes podem ser resolvidas "). A metodologia tornou-se classificada como um candidato admissível para os projetos na categoria "desenvolvimento rápido de aplicações", e está programado para ser usado novamente no futuro próximo.

Aqui está o relatório, cortado para caber dentro deste capítulo.

O Relatório de Campo

Sumário executivo

Este relatório descreve um projeto piloto utilizando a metodologia Crystal Clear. O relatório representa parte de um corpo de conhecimento sobre metodologias ágeis construídas na Thales Pesquisa e Tecnologia (TRT UK), como parte da sua actividade Pequenos Sistemas de engenharia de software. si Crystal Clear foi desenvolvido pelo methodologist Alistair Cockburn.

Crystal Clear é uma metodologia Agile. 'Agile' é um termo genérico para metodologias como XP, DSDM e Scrum. Em geral, as metodologias ágeis visam reduzir o risco de construir a coisa errada, e para entregar valor o mais cedo possível. A pesquisa bibliográfica das metodologias ágeis foi realizado em 2003, e foi decidido que Crystal Clear foi o mais susceptível de ser útil na Thales 60. Crystal Clear foi selecionado porque ele enfatiza

- **Eficiência. Crystal Clear visa minimizar o desperdício, embora centrado a equipas sobre as características importantes.**
- **Habitabilidade. Crystal Clear é projetado para ser confortável de usar, para que as equipas não se sentem a necessidade de evitar que se lhe segue.**
- **Segurança. A metodologia tem como objetivo aumentar a probabilidade de entrega bem sucedida.**

Crystal Clear é destinado a pequenos projetos, de baixa criticidade. Comparado com o XP (que é a metodologia Agile mais conhecido), Crystal Clear é mais tolerante com diferentes formas de trabalhar dos indivíduos, mas inclui atividades de planificação mais fortes.

Corremos um projeto-piloto, a fim de avaliar a metodologia. Nosso objetivo foi testar a metodologia contra as reivindicações feitas por ele, e para determinar quaisquer problemas que possam impedir a adopção. Projeto 'CamCal' foi escolhido como o piloto porque era pequeno e não crítico (que é o tipo de projeto a metodologia foi projetado para), e porque seus membros estavam dispostos a fazer parte do julgamento. Infelizmente, o projeto foi interrompido antes de sua data de término planejada, por razões fora do âmbito do projeto piloto. O projeto ainda conseguiu entregar um produto para a satisfação de seus clientes.

Nossas conclusões estão resumidas a seguir. A metodologia cumpriu os seus objectivos de ser eficiente, habitável e segura:

- **Eficiência. métricas de desempenho mostraram alta produtividade.**
- **Habitabilidade. A equipas ficaria feliz em usar a metodologia de novo.**
- **Segurança. O projeto foi convergindo para a sua data final acordada até que foi reduzido. O projeto ainda criado um produto utilizável.**

60 Esta era a visão de Stephen. Não é uma posição Thales corporativa.

As seguintes questões precisam de atenção na adoção da metodologia:

- Ela pode exigir papéis para mudar estabelecida
- Ele desafia as práticas de documentação estabelecida
- Ela exige um contrato amplo de escopo

A equipa TRT UK Quality Assurance realizou uma avaliação informal do projeto a partir da perspectiva da ISO 9001 e TickIT, concluindo que a metodologia poderia ser feita em conformidade com essas normas, com a adição de algumas práticas extras e registros. A equipas de QA propôs que Crystal Clear ser incorporada no Sistema de Gestão da Qualidade no TRT Reino Unido, e propôs novos estudos utilizando a metodologia.

Uma análise da metodologia a partir da perspectiva CMMI está em curso no momento da escrita. Temos como objectivo a julgamento a metodologia em um projeto com mais desenvolvedores.

Introdução

O projeto desenvolveu um programa para uso em conexão com um sistema de visão computacional chamado ZYX. Nosso projeto piloto, CamCal, desenvolveu uma ferramenta de calibração da câmara para ZYX. O objetivo da calibração da câmara é informar ZYX de aonde a câmera é, e aonde ele está apontando. Esta informação é codificada em uma matriz de calibração que mapeia coordenadas mundo 3D, para coordenadas imagem 2D. A matriz é necessário por ZYX a fim de compreender o que vê.

O ponto de partida foi uma ferramenta de calibração da câmara protótipo que já tinha sido desenvolvido em MATLAB. O objetivo do projeto era substituir o protótipo com algo menos frágil, mais adequado para apresentar aos clientes, e mais adequado para uso em sites de clientes. Decidimos escrever a nossa nova versão em Java, com alguns dos componentes matemáticos implementados em MATLAB.

CamCal era um projeto pequeno, com apenas 1_ desenvolvedores e um custo total de pouco mais de seis meses homem. O trabalho foi financiado internamente pelo TRT Reino Unido e não havia nenhum cliente externo imediato.

Cronologia do projeto

As datas-chave do projeto são mostrados na Erro! Fonte de referência não encontrada. abaixo. Para manter esta secção razoavelmente curto, iremos apresentar apenas os eventos mais significativos sobre o projeto. Estes incluem eventos em que houve uma alteração dos requisitos ou a programação.

O projeto envolveu as seguintes pessoas. Com a exceção de Stephen, que é o autor do presente documento, os seus nomes foram alterados.

- Sam, o gerente de projeto a cargo do projeto mais amplo de que CamCal era uma parte

- Liam, um consultor de visão computacional

- **Stephen, um engenheiro com uma programação / algoritmos de fundo**
- **Desmond, um engenheiro com um fundo Java**

protótipo de Liam

Liam concebeu a idéia de uma ferramenta de calibração da câmara. Ele escreveu um protótipo, principalmente em seu próprio tempo. O protótipo inclui todas as características-chave necessárias, mas era frágil e não particularmente fácil de usar. O que era necessário era uma ferramenta escrita para os padrões profissionais que poderíamos oferecer aos clientes.

Proposta

Sam e Liam começou a trabalhar em uma proposta de projeto de calibração da câmara. Eles precisariam de pelo menos um engenheiro de software. Stephen seria liberado para trabalhar no projeto de calibração de câmera em uma base a tempo parcial, desde que o projeto utilizar uma metodologia Agile.

Um documento proposta foi escrito que estabeleceu requisitos iniciais do projeto, a abordagem de desenvolvimento, uma estrutura de divisão de trabalho inicial e uma estimativa para o trabalho. A estimativa era de 105 dias de esforço. A atribuição inicial dos papéis foi mostrado abaixo.

Nome	Roles
Sam	Patrocinador
Stephen	Designer-chefe, Coordenador, escritor, Tester
Liam	Utilizador, Designer, Business Expert, Escritor, Tester

Figura 8-1. Atribuição inicial dos papéis

O documento de proposta foi posta à alta administração e aceito. O projeto teve início em 18 de agosto º 2003.

O documento de proposta foi baseada em uma versão inicial (Jan 2002) de Crystal Clear. Isto não inclui projeto atividades de start-up. Mas ficou claro que havia questões que precisávamos para resolver antes que pudéssemos começar o desenvolvimento iterativo. Então, nós emprestado a idéia de uma 'fase de exploração' do XP. Esta foi a primeira fase do projeto e é descrito a seguir.

		Evento	Esforço (dias) Da	tas
Startup	projeto	protótipo de Liam		Dezembro 2002 a fevereiro 2003
		Proposta		junho 2003

	Exploração		Seg 18/8/03 Fri 5/9/03
	Desmond une o projeto		ter 9/9/03
	Primeira Reunião de planificação		Qua 10/9/03
incremento1 e	Desenvolvimento		Qui 11/9/03 a Sex 17/10/03
	Visualizando		Sex 17/10/03
	workshop de reflexão		Seg 20/10/03
incremento2 ar	Reunião de planificação		Seg 20/10/03
	Desenvolvimento		Seg 20/10/03 a 7/11/03 Fri
	Visualizando		Sex 7/11/03
	workshop de reflexão		Qui 13/11/03
incremento 3 m	Reunião de planificação		Qui 13/11/03
	Desenvolvimento		Seg 10/11/03 a 5/12/03 Fri
	Visualizando		Mon 8/12/03
	oficina de reflexão		Mon 8/12/03
incremento 4 consolidarpara fimde ano	Reunião de planificação		Mon 8/12/03
	Desenvolvimento		Ter 9/12/03 a 23/12/03 Tue
	Visualizando		Ter 23/12/03
	oficina de reflexão		Ter 23/12/03
Increment	Reunião de planificação		Mon 5/1/04
	Desenvolvimento		Mon 5/1/04 a 23/1/04 Fri

	Visualizando		Seg 26/1/04
	oficina de reflexão		Seg 26/1/04

Figura 8-2. cronologia do projeto

Exploração

A 'Fase de Exploração' era uma tarefa longa de duas semanas em que Stephen e Liam capturado requisitos mais detalhados, investigou as principais questões técnicas, e firmou-se as estimativas 61.

O documento de proposta deu um conjunto de requisitos de alto nível, mas era necessário mais detalhes. Stephen e Liam trabalharam juntos à carne para fora os detalhes, com Liam (como utilizador) sendo a principal fonte de requisitos. Capturaram os requisitos utilizando um quadro de impressão, desenho vários projetos de screen-shots tal como o mostrado abaixo (Figura 8-3).

As impressões foram contados e mantidos juntos numa pasta (a impressão em baixo é o número 10). Eles também começaram um documento de requisitos.

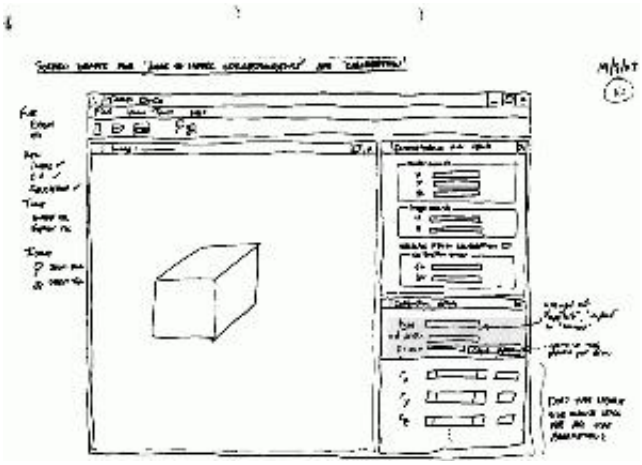


Figura 8-3. Um exemplo de impressão branco bordo

O documento de proposta tinha deixado em aberto a questão de qual idioma o software estava a ser escrito. Isso precisa ser resolvido rapidamente, porque ela afetou a composição da equipas. Depois de algumas investigações, decidiu-se usar Java para a GUI, e MATLAB para a matemática subjacente. O código-fonte MATLAB iria ser compilados em uma DLL compatível com o Java Native Interface.

61 Os leitores vão reconhecer isso como semelhante ao Exploratórios 360 ° estratégia descrita no Capítulo 3.

Crystal Clear espera que pelo menos um membro da equipas que está bem familiarizado com a tecnologia, mas nem Liam nem Stephen tinha usado Java antes. Por isso pedimos para outro membro da equipas, e recebeu Desmond, uma forte desenvolvedor Java. Desmond mudou-se para sentar-se com Liam e Stephen na mesma baía.

No final da fase de exploração, as funções do projeto foram como mostrado na Erro! Reference source not found .. Desmond foi atribuído ao projeto 100%. Stephen tinha um compromisso com outro projeto de modo foi alocado 50%. Os outros

membros da equipas foram chamados quando necessário. Stephen é mostrado como Designer de chumbo porque esse era o plano original. Mas Desmond tinha o conjunto de habilidades mais relevantes, e ficou claro que ele levaria o trabalho de desenvolvimento uma vez que ele foi até a velocidade com o projeto.

Nome	Fundo (parcial)	Papel (s)
Sam	Gestor de projeto	Patrocinador
Liam	Consultor técnico	Do utilizador , Business Expert, Escritor, Tester
Stephen (50%)	engenheiro de software, matemática engenharia	Designer-chefe , Coordenador, Escritor, Tester
Desmond (100%)	engenheiro de software (especialista em Java)	estilista , Tester

Figura 8-4. As funções do projeto no final de exploração

Primeira reunião de planificação

Foi realizada uma reunião de planificação usando o planificação do projeto 'jam session' técnica sugerida em CC.Sam, Desmond e Stephen sentou-se em volta de uma mesa e contribuiu cartões de tarefas. Um exemplo de cartão de tarefa é mostrado abaixo, na Figura 8-5. Cada carta tarefa tinha um título, uma breve descrição, o nome da pessoa, e uma estimativa em homem dias. A descrição não precisa ser muito longo, nem compreensível para qualquer pessoa fora do projeto. A estimativa foi no canto superior direito no homem dias. A equipas adotou a convenção que a estimativa inclui qualquer esforço de teste de unidade associado.

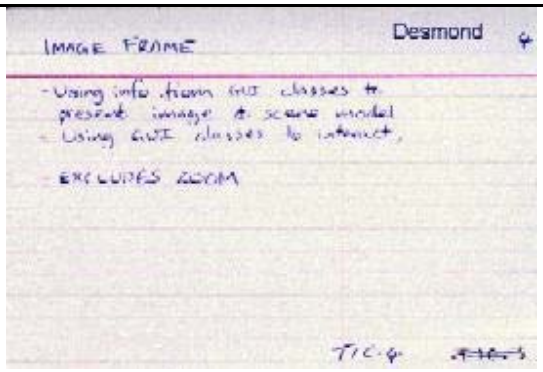


Figura 8-5. Um exemplo cartão de tarefa

As principais tarefas foram para Desmond, como a pessoa com o mais forte experiência em Java. Nós espalhamos os cartões de tarefas sobre a mesa e concordaram uma ordenação que fazia sentido. (A programação que 'faz sentido' é aquela que maximiza as possibilidades de alcançar os objectivos do projeto, como indicado na declaração de missão). Em seguida, deu a cada um cartão de um rótulo (na parte inferior do cartão) para que pudéssemos reconstruir a ordenação. Após a reunião, as tarefas foram copiados para o Microsoft Project, que é a ferramenta de agendamento padrão no TRT Reino Unido.

Na verdade, a maioria dos cartões de tarefa foram criadas antes da reunião, por Stephen e Liam, que tinha passado por este processo já. Desmond essencialmente revisitou as descrições de tarefas, escolhas tecnológicas, e estimativas. No entanto, a primeira reunião de planificação foi um ponto-chave no projeto. O projeto teve uma direção clara, e todos tinham contribuído.

Somando os totais sobre os cartões de tarefas, chegamos a uma estimativa melhorada para o restante do projeto: 134 dias-homem. Adicionando o custo de exploração (já gasto) deu 154 dias-homem. Esta foi maior do que a estimativa no documento de proposta, mas o patrocinador estava feliz para o trabalho para continuar na base do cronograma tínhamos acordado.

Figura 8-6. Uma extremidade da tabela que estava coberto de cartões de tarefas

Incremento de 1: Andando Skeleton

O objetivo do primeiro incremento foi criar um aplicativo esquelético incorporando as principais características arquitetônicas e algumas funcionalidades do núcleo. Em Crystal Clear linguagem isso é chamado de 'esqueleto andando'. As principais tarefas seria a criação do modelo de objeto, e implementação do software para projetar um modelo de cena na tela.

Primeiro tivemos que selecionar e configurar as nossas ferramentas. No TRT Reino Unido, projetos decidir principalmente por si quais as ferramentas que eles vão usar. As ferramentas foram escolhidos pelos membros da equipas, especialmente Desmond, com base na sua experiência em outros projetos. As ferramentas foram os seguintes.

- **ferramenta de design. Juntos 5.5 foi seleccionado. Isso gera código Java esquelético a partir de diagramas UML. As alterações posteriores do software são automaticamente reflectida na representação UML.**

- **ferramentas de programação. NetBeans 3.5 foi escolhido como o nosso Java IDE, em execução em J2SE 1.4.2. MATLAB 6.5 já havia sido escolhido para o software de calibração, com o Microsoft Visual C ++ para a DLL 'lógica cola' que ligaria o MATLAB compilado para Java.**

- **ferramenta de teste de unidade. Selecionamos JUnit.**
- **ferramenta de controle de versão. Usamos Microsoft SourceSafe 6.0, que é padrão no TRT Reino Unido.**

Uma vez que as ferramentas foram selecionados e configurado, Desmond começou a criar a arquitetura do software, discutindo questões de design com Stephen usando a impressão

quadro branco. O aplicativo foi projetado utilizando o paradigma do 'Model-View-Controller'.

Neste ponto, precisamos dizer um pouco mais sobre o aplicativo. A aplicação utiliza um modelo de cena 3D, que descreve os objectos na região observada pela câmara. O modelo de cena consiste de um conjunto de pontos 3D, que são unidas em conjunto para formar as arestas dos objectos. O modelo de cena é projetada sobre a tela 2D de acordo com a localização e orientação de uma câmera virtual. A imagem formada pela câmera virtual é chamado de "modelo de cena projetada.

Durante incremento de 1, Stephen escreveu o código Java para transformar o modelo 3D cena em coordenadas de tela 2D.

Aqui, houve uma mudança de planos. O cronograma original tinha chamado para este software a ser escrito em MATLAB, compilado para DLL e acessado pela JNI. Logo ficou claro que não estávamos indo para atingir esse nível de integração no tempo disponível para incremento de 1. Então, Stephen escreveu o código em Java em vez disso: desta forma, Desmond teria algo para integrar dentro incremento de 1. Se o software provou ser demasiado lento, pode ser reescrita em MATLAB mais tarde. (MATLAB é otimizado para operações de matriz, então esperamos um desempenho mais elevado). Stephen e Desmond re-planejado seu trabalho através da realização de uma breve reunião de planificação.

Simultaneamente, a equipas continuou a aperfeiçoar os requisitos, com Liam e Stephen trabalhar em conjunto para produzir rascunhos de tela atualizados. A maioria dos requisitos discutidos nesta fase em causa questões de usabilidade. Liam fez um pedido de completamente nova funcionalidade (remoção de linha oculta), e este foi recusado porque foi encontrado para complicar o projeto demais. Sabíamos que a principal preocupação de Sam era para entregar a tempo, ao invés de entregar o produto mais abrangente.

Até o final de Incremento 1, a equipas havia conseguido criar o esqueleto ambulante. O modelo cena foi apenas um manequim, que consiste em algumas formas geométricas (uma caixa, um prisma, etc). A aplicação final seria usar um modelo de cena que representa a região que está sendo visto, neste caso, parte de uma estação ferroviária. O utilizador pode clicar e arrastar o ícone da câmera em torno da vista de planta, e o aplicativo automaticamente atualizou o modelo de cena projetada.

Foi apenas um início, mas suficiente para demonstrar algumas funcionalidades núcleo. Para o final do incremento, Sam deu ao time da sua 'declaração de missão'. Este é um dos produtos de trabalho exigidos pela Crystal Clear. É o único produto de trabalho perguntou do Patrocinador.

Camera Missão Calibração

A missão do projeto calibração da câmera é:

- Desenvolver uma ferramenta de software de fácil manutenção, simples de usar, portátil e extensível que permitirá vídeo inteligente instaladores e reparadores para sistema de vigilância rapidamente,

precisão e eficiência calibrar suas câmeras do sistema com respeito a uma pré-definido modelo de cena tridimensional.

- Estabelecer uma base para o desenvolvimento de um conjunto de ferramentas futuro que pode gerenciar todos os aspectos das atividades de instalação do sistema de vigilância de vídeo e oficinas de reparação inteligentes. Tal conjunto de ferramentas irá incluir a ferramenta de calibração de câmera, e irá adicionar capacidades de criação de modelo de cena. Ele vai ser capaz de crescer com o crescimento da tecnologia do mercado de vigilância por vídeo.

- Desenvolver e julgamento do uso de métodos e de novos processos de desenvolvimento de software, com vista a melhorar o processo de desenvolvimento de software utilizado dentro Thales, para pequenas e médias projetos de software.

As prioridades de desenvolvimento são: sacrificar os outros para isso:
Concluído até final de Janeiro de 2004.

	Garantir a precisão e qualidade
Manter, se possível:	Usabilidade, potencial para crescer em uma ferramenta mais extensa
Sacrificar estes primeiros:	velocidade de execução, portabilidade, interfaces adicionais, armazenamento
	formato interno.

Figura 8-7. declaração de missão CamCal Calibração

No final do incremento de 1 realizamos uma visão e uma oficina de reflexão.

Vendo 1

A reunião Visualizando foi realizada em Sex 17/10/03 no final do incremento 1. Desmond demonstrou a aplicação de Liam e Sam. O encontro foi cordial, com Liam e Sam tomando um grande interesse nos detalhes do produto. Várias questões foram levantadas em conexão com o software e estes foram registrados no quadro de impressão. A impressão de quadro é mostrado abaixo na Figura 8-8. No final da visualização a equipas teve uma vista do status do produto que foi compartilhado, completa e razoavelmente positivo.

Em CamCal, optamos por ter uma visualização no final de cada incremento. Isso é mais do que Crystal Clear demandas, a metodologia chamadas apenas para um mínimo de dois por liberação.

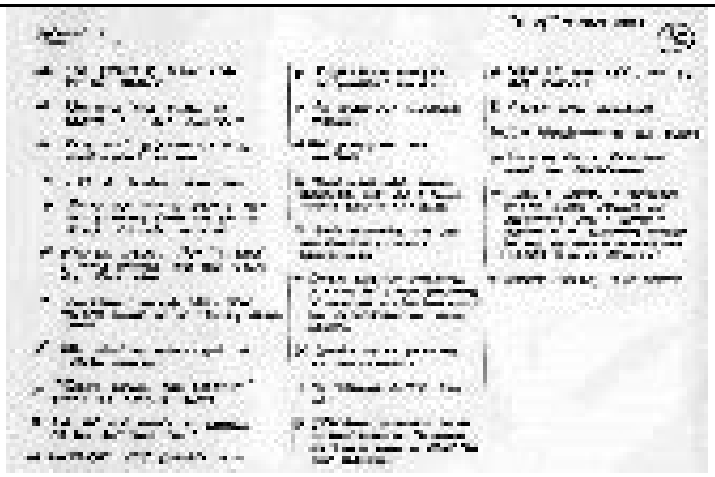


Figura 8-8. impressão Whiteboard de Vendo 1

Workshop de reflexão 1

No final do incremento de 1, a equipas realizou uma reflexão Workshop. Seu objetivo foi identificar quais os aspectos do processo estavam trabalhando e que precisava ser mudado. As notas do workshop reflexão são mostrados na impressão quadro abaixo. Os principais pontos foram

- **Tudo para passar um dia lendo CC. A equipas não tinha sido treinado no uso da metodologia. O único material de treinamento que estava disponível era o manual sobre CC, por isso foi sugerido e acordado que cada membro da equipas deve passar um dia lê-lo.**

- **Desmond como chumbo Designer. Desmond tinha vindo a fazer as principais decisões técnicas, por causa de sua experiência com Java. Stephen tinha acabado de ser agindo como 'campeão processo'. Propôs-se a reconhecer esta situação nomeando Desmond o Designer de chumbo. Desmond não tinha certeza, mas concordou em pensar nisso.**

- **'Guarde este' declarações. A equipas descobriu muito sobre a metodologia que eles queriam manter. Em particular, os membros da equipas estavam trabalhando bem juntos. Vários dos membros da equipas tinham trabalhado juntos antes em projetos que tinham sido executados de forma diferente: eles estavam achando mais fácil trabalhar juntos neste projeto.**

- **Ausência de membros da equipas. Além de Desmond, todos os membros da equipas tinha compromissos significativos para outros projetos. A equipas reconheceu isso como um problema. Métodos ágeis dependem de conhecimento tácito, por isso, são vulneráveis se as pessoas não estão presentes.**

- **Incompatibilidade com os procedimentos TRT Reino Unido. Sam estava desconfortável com o fato de que ele era formalmente responsável pelo projeto, mas não no controle de sua programação 62.**

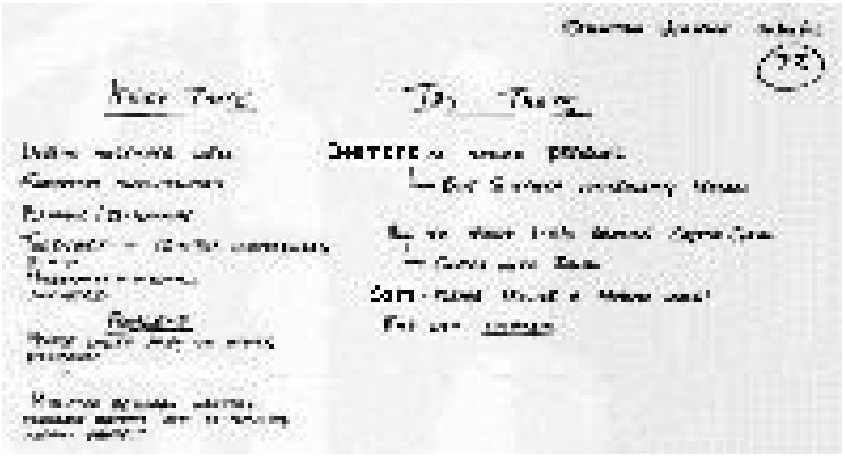


Figura 8-9. impressão Whiteboard da reflexão oficina 1

Incremento 2: CP Entrada e edição

O objectivo para incremento de 2 era permitir ao utilizador inserir e editar 'Pairs correspondência' (CPs). Um par correspondência especifica uma associação entre um vértice no modelo de cena, que é tridimensional, e um ponto na imagem, o qual é bidimensional. CPs são inseridos no CamCal por operações de mouse. O utilizador clica em um ponto na imagem, e arrasta para o vértice pretendido no modelo de cena projetada. Os CPs são finalmente usado como entrada para o algoritmo de calibração, que calcula a matriz de calibração necessário por ZYX.

Incremento 2 começou com uma reunião de planificação (de segunda 20/10/03) em que o cronograma do projeto foi reconstruída com base na situação atual. A principal mudança em relação ao plano original era que Stephen não estaria disponível para incremento 2. Ele tinha um compromisso com outro projeto e não teria tempo. Depois de discutir as opções, a decisão foi tomada para atrasar a sua contribuição até incremento! 3.

Na nova programação, algum tempo foi alocado para resolver os problemas capturados pelo Vendo 1 (Erro! Fonte de referência não encontrada).: mas a prioridade era a avançar com os aspectos de alto risco do projeto, tais como a calibração e a integração com ZYX.

Até o final do incremento, a possibilidade de adicionar e editar pares de correspondência estava pronto para demonstração.

É **utilizado um modelo de cena real, em vez de formas geométricas apenas.**

62 (Vamos ver discussão sobre este ponto no final do relatório de campo.)

Nós realizada outra sessão de visualização, e outra oficina de reflexão. A visualização gerado mais 'problemas', que foram gravadas em uma cópia impressa quadro branco. Novamente, a equipas tinha conseguido uma visão compartilhada do status do produto.

Este workshop reflexão gerado menos pontos do que o primeiro tinha. Os principais resultados do workshop reflexão foram

- No workshop reflexão anterior, os membros da equipas tinha concordado em ler sobre a metodologia, mas eles não tinham encontrado o tempo. Desmond concordou em fazer isso em incremento de 3.

- O projeto não estava se encontrando certas partes da metodologia. As avaliações pelos pares não tivesse acontecido. A lista de riscos não tinha sido mantida. O documento de requisitos não haviam sido firmadas acima. Concordamos em resolver estes em incremento de 3.

Incremento 3: Calibração

O incremento começou com uma reunião de planificação. As tarefas para esse incremento estão resumidas abaixo. Eles foram todas concluídas até o final do incremento.

- revisão do código de pares
- Implementar o algoritmo de calibração em MATLAB, e integrar isso com Java
- Implementar instalações GUI apoio calibração
- teste do sistema inicial
- Desmond ler CC

A integração entre MATLAB e Java tinha sido considerada uma área de risco, por isso foi um alívio para alcançá-lo.

Nós tentamos postar os cartões de tarefas para o incremento em um quadro branco na área de trabalho da equipas. Isto provou ser muito eficaz em comunicar o cronograma, e nós manteve a prática para o restante do projeto.

Além destas tarefas agendadas, os requisitos continuou a ser refinado em uma base contínua. exigência original de Liam tinha sido que o apoio CamCal dois algoritmos de calibração particulares. Mas quando a integração foi discutido, aprendemos que apenas um algoritmo era compatível com o nosso sistema de destino. Uma vez que o objetivo do projeto era desenvolver uma ferramenta de calibração para esse sistema e a equipas teve um calendário apertado, a decisão foi tomada para omitir o segundo algoritmo.

Stephen firmado até o documento de requisitos, e se tivesse revisado por Liam e Desmond. Também houve mais discussão sobre o comportamento detalhado GUI. Vamos relacionar alguns detalhes da discussão, para dar uma sensação para o tipo de decisões que estavam ocorrendo.

Durante Incremento 1, Stephen e Liam tinha esboçado uma interface de utilizador que foi assistente-like, para que o utilizador seria guiado através das atividades que a calibração envolve. Cada página do assistente teria painéis de tamanho fixo e posição. Desmond, que era mais experiente em design de GUI, senti que isso seria demasiado

restritiva e favoreceu um design mais flexível, dando ao utilizador a capacidade de se mover, redimensionar e mostrar / ocultar as janelas. Ele tinha implementado o software para que este aspecto da sua aparência poderia ser facilmente alterado. A dificuldade era que, dando ao utilizador esse nível de controle, a natureza do tipo assistente do aplicativo foi perdido, e ele pode não ser óbvio para o utilizador o que ele precisava fazer.

O que era necessário era uma resolução para a questão de como o GUI deve se comportar. Na reunião de planificação, Desmond tinha tomado uma tarefa para resolver a questão fora. Desmond criado um projeto no qual o utilizador pode selecionar entre modos de exibição usando os botões de rádio. As vistas eram

- **alinhamento manual. Aqui, o modelo de cena projetada reflete a posição e orientação de uma câmera virtual que o utilizador poderia controlar.**
- **calibração algorítmica. Aqui, o modelo de cena projetada foi formado usando o algoritmo de calibração.**
- **calibração xxxxx. Esta foi uma opção manequim. Desmond deixou-a no lugar para mostrar que a interface do utilizador pode ser estendido para conter mais de um algoritmo de calibração.**

Liam sentida este arranjo ainda não foi ideal, porque os controlos para a vista 'manual do alinhamento' eram visíveis quando a primeira calibração foi seleccionado.

Nós apresentamos o sistema em uma visualização em Mon 8/12/03. Nesta fase, a maior parte da funcionalidade do núcleo da aplicação estava presente. Pode-se carregar uma imagem e um modelo de cena, adicionar pares de correspondência, e executar o algoritmo de calibração. O algoritmo de calibração encontra uma transformação que se encaixa no modelo de cena na imagem 2D, de modo que a projecção do fio-frame do modelo cena está alinhada com objectos na imagem.

Por esta visualização, um representante QA tinha sido convidado. Este foi o primeiro envolvimento QA com o processo. Posteriormente, um representante QA foi convidado para todas as reuniões sobre o projeto. [A perspectiva QA é apresentada separadamente, seguindo este relatório de campo.]

Incrementar 4: Consolidar para o final do ano

O principal objectivo para este incremento foi de trazer o projeto para um estado aonde ele poderia ser fechado relativamente indolor. Havia uma possibilidade de que problemas fora do escopo do projeto pode levar a que fosse fechado no final do incremento (que coincidiu com o final do ano). Na verdade, esta exigência não afetou o planificação muito. As principais tarefas que precisava ser feito para a nossa meta de final de Janeiro de 2004, precisava ser feito de qualquer maneira. As tarefas para esse incremento estão resumidas abaixo. Todos eram concluída até o final do incremento.

- **Criar um CD do aplicativo, incluindo uma ferramenta de instalação**
- **Integrar com ZYX. Isto envolveu tendo CamCal criar um arquivo de calibração e teste que ZYX poderia fazer uso dele.**
- **Iniciar um Manual do Utilizador. (Não poderia ser terminado até que a interface de utilizador estava completa)**

- As revisões de código e ações associadas.

Neste ponto, o projeto tinha uma página de 'problemas' de cada visualização. Liam priorizados eles, e eles foram copiados para uma lista de problemas que Desmond tinha criado no Microsoft Outlook. Usando Outlook significava que todos nós teria acesso instantâneo à lista mais recente.

O incremento encerrado com uma visão e um Seminário de Reflexão. Por esta visualização, corremos o software diretamente do CD de instalação, para demonstrar que isso funcionou. Apenas alguns pontos foram levantados na visualização, todo mundo sabia que não havia tempo para grandes mudanças. O workshop reflexão foi tranquila, bem como, sem ninguém sugerir qualquer alteração no processo.

Incremento de 5: Complete

O período de férias de Natal forneceu uma pausa natural entre incrementos, eo quinto incremento começou na segunda-feira 5/1/04.

No final de 2003, houve uma decisão, feita fora do escopo do projeto, que CamCal deve ser concluída mais barato no início de 2004. Para isso, foi tomada a decisão de omitir os testes de aceitação e revisões de código que tinha anteriormente foi agendada. A justificativa era que o software já havia passado várias atividades de teste e de revisão, e este tinha criado um grau de confiança no código. As principais tarefas para o incremento foram

- A lista de problemas priorizados. Após negociações, foi acordado que Desmond iria trabalhar para baixo na lista questões, abrangendo todas as questões até que ele correu para fora de tempo, ou até um certo ponto na lista de prioridades foi alcançado, o que viesse primeiro.

- Liam para terminar o manual do utilizador
- Desmond e Stephen para escrever um manual de manutenção No final do incremento, que realizou uma visualização final. Comparado com o incremento anterior, que era geralmente um pouco mais arrumado e mais polido. Este foi o resultado do trabalho de Desmond abordar os itens na lista de problemas priorizados. Sam aceitou a aplicação imediata para demonstrar aos clientes, e pediu que ele ser instalado no sistema de apresentação ZYX.

Nós realizou um workshop reflexão final. Isto se transformou em uma discussão geral sobre como a metodologia foi geral. vista dos membros da equipas foram geralmente muito positiva. O único problema substancial levantada foi o ajuste com os procedimentos TRT Reino Unido. Se isso pode ser resolvido satisfatoriamente, então todos os participantes ficaria feliz em trabalhar desta forma novamente.

(Na verdade, não é um procedimento no TRT Reino Unido para abordar SGQ problemas deste tipo: qualquer funcionário pode levantar sugestões para melhoria de processos).

A conformidade com Crystal Clear

Nesta seção, resumimos a medida em que o nosso projeto piloto foi consistente com Crystal Clear. Nós avaliamos a conformidade em relação aos seguintes aspectos da Crystal Clear: as propriedades, os papéis, o ciclo de vida, e os produtos de trabalho.

- **propriedades**

  As propriedades CC foram alcançados.

- **Roles**

  O projeto piloto papéis alocados para as pessoas como Crystal Clear pergunta. Havia alguns desvios do ideal:

  O papel 'Chumbo Designer' teve que ser compartilhada entre Stephen e Desmond. Stephen tomou o aspecto metodologia do papel, e Desmond tomou a liderança técnica. Esta divisão foi necessária porque o papel Designer de chumbo exige conhecimento tanto das questões técnicas e metodologia. Enquanto Desmond foi a 'Designer de chumbo', no sentido literal, ele não tinha sido treinado na metodologia e assim por Stephen teve que tomar essa parte do papel. Esta divisão de responsabilidades causou alguma confusão em lugares, eo projeto teria funcionado melhor se o papel foi tomada por uma única pessoa.

  Houve um desvio no papel de utilizador. O produto tinha duas finalidades: (1) como uma ferramenta de demonstração para ser usado por figuras importantes no TRT Reino Unido, e (2) como uma ferramenta de instalação a ser utilizado por engenheiros de instalação. Liam é um verdadeiro utilizador de primeira categoria. Nós não tínhamos um utilizador na última categoria.

- **Ciclo da vida**

  O projeto piloto inicialmente utilizado um rascunho inicial da metodologia, que não incluem o projeto as atividades de start-up cristalina. Estes foram, por conseguinte, omitida. Em outros aspectos, o projeto seguiu o ciclo de vida Crystal Clear. Nós observar o seguinte:

  a) A metodologia pede libertação frequente de software aos utilizadores, de modo a obter feedback. Nosso projeto piloto conseguido que por ter um utilizador na equipas. O utilizador foi dada uma cópia atualizada do programa de vez em quando.

  b) Crystal Clear exige um mínimo de duas sessões por libertação. Tivemos mais visões do que esta, depois de ter tomado a decisão de ter uma visualização no final de cada incremento. Todas as sessões foram úteis, mas as visões posteriores foram menos úteis, porque o produto tinha mudado relativamente pouco entre incrementos.

- **produtos de trabalho**

  O projeto piloto criado a maioria dos produtos de trabalho necessários, além do seguinte

  1. Como não havia apenas um lançamento, o 'seqüência de liberação' e 'liberação agendar' reduzida para um único final de data.

2. O produto 'código de migração' trabalho é para aplicações de banco de dados e não foi aplicável para o nosso projeto piloto.

3. Os testes de aceitação foram removidos a partir do cronograma quando o projeto foi reduzido, por isso não houve resultados dos testes de aceitação.

Análise

Desempenho contra estimativas

As prioridades para o projeto-piloto, tal como indicado na declaração de missão foram

Sacrificar os outros para isso:	**Concluído até final de Janeiro de 2004.**
	Garantir a precisão e qualidade
Manter, se possível:	**Usabilidade, potencial para crescer em uma ferramenta mais extensa**

O projeto foi dirigido, de modo a responder a estas prioridades. Note-se que o objetivo principal era a data final, não o custo final. Assim, sempre que a data final ou usabilidade parecia ameaçada, a equipas reagiu por gastar esforço para resolver o problema. A equipas foi capaz de fazer isso porque apenas um dos seus membros foi trabalhar no projeto em tempo integral. Os outros tinham alguma flexibilidade em quanto tempo eles comprometidos com o projeto.

Isto reflecte-se no gráfico da estimativa contra esforço real mostrado na Figura 8-10 abaixo. As estimativas aumentar gradualmente a partir da primeira reunião de planificação, para o quarto. A quinta reunião de planificação mostra uma estimativa reduzida que reflete a decisão de reduzir o projeto. (O projeto reduzida conheceu a necessidade do negócio subjacente, que era para um produto demonstrável, não uma entrega um).

A fase de proposta não foi conduzida de acordo com o processo Crystal Clear, por isso não faz parte da nossa experiência. Criou-se uma estimativa que foi encontrado, dentro de algumas semanas de começar o projeto, para ser uma subestimação considerável. A principal razão para o problema foi que a fase de proposta não incluir o tempo suficiente para determinar a tecnologia que seria utilizada ou o pessoal que faria o projeto.

Na verdade, os valores indicados entre Erro! Fonte de referência não encontrada. Não foram computados até depois do projeto. Em retrospectiva nós sentimos que teria sido vantajoso ter calculado os números em tempo real.

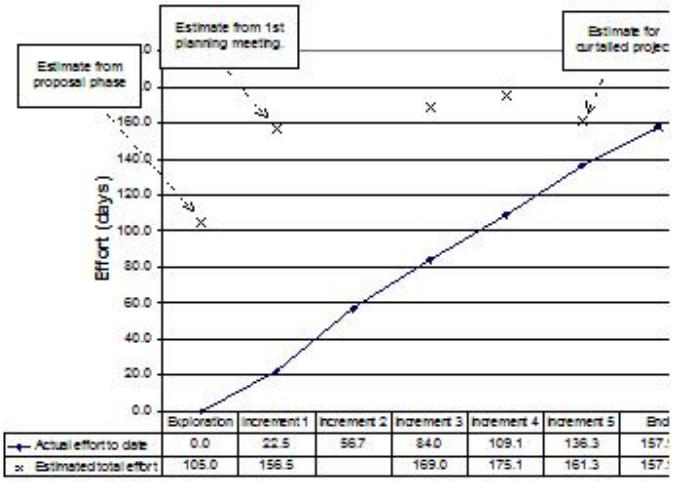


Figura 8-10. Estimativa Precisão (valores reais e estimativas no começar de cada incremento)

Análise do ponto de vista ISO 9001

[O relatório campo continha um trecho da análise do auditor. A mais longo trecho de seu relatório é apresentado em separado, seguindo este relatório de campo.]

conclusões

Esta secção irá apresentar a nossa avaliação de Crystal Clear. Nós avaliamos a metodologia contra as reivindicações que são feitas para ele. Apresentamos também uma série de questões em que é necessário cuidado na utilização da metodologia.

Avaliação de Crystal Clear contra suas reivindicações

As reivindicações feitas para Crystal Clear é que ela é eficiente, habitável e segura. Na base do nosso projeto-piloto, que apoiar estas alegações.

- **Eficiência**

A métrica de produtividade final para o projeto foi 81,7 linhas de código não-comentário (código do sistema mais o código de teste de unidade) por dia. Esta métrica foi calculado dividindo-se o número total de linhas de código geradas, pelo esforço total gasto, incluindo todo o esforço não-codificante. [Nota: o código de teste unitário foi de 26,4% do código Java.]

O projeto não incluem testes de aceitação, mas isso não teria mudado os valores globais muito. testes de aceitação provavelmente teria encontrado alguns pequenos erros que teria, então, corrigidos, mas no geral, nós sabíamos que o Utilizador e Patrocínio já estavam satisfeitos com o produto, e já tinha sido objecto de vários testes.

Este alto nível de produtividade surgiu, em primeiro lugar, das habilidades do designer (Desmond). A metodologia ajudou, mantendo o desenvolvimento focado, e reduzindo o desperdício. 'Resíduo', neste contexto, significa qualquer coisa que não contribui para o produto final.

- habitabilidade

Todos os membros da equipas ficaria feliz em usar essa metodologia novamente. Vários comentários positivos foram recebidos da equipas, incluindo:

Liam (User). “No geral, tomando parte foi uma experiência agradável, e que eu ficaria feliz em repetir”

Desmond (Designer). “Normalmente, eu me senti muito focado e energizado ... Eu senti geralmente no controle”.

Sam (Patrocinador). “As comunicações eram excelente ... a equipas trabalhou bem juntos .... o software foi entregue em uma condição de trabalho satisfatório “.

Outros comentários da equipas são apresentados mais adiante. Sam tinha algumas preocupações sobre o ajuste entre as definições de função em Crystal Clear e os esperados pelo TRT Reino Unido. aprovação da metodologia de Sam era contingente sobre esta questão a ser resolvido satisfatoriamente.

- Segurança

A metodologia de 'seguro' é aquele que aumenta a probabilidade do projeto ser bem sucedido. O projeto-piloto bem sucedido na medida em que criou um produto que satisfez o Patrocinador. Além disso, vários aspectos do projeto foram comprovadamente seguro: os planos eram sempre significativas, questões difíceis foram resolvidos em tempo hábil, e a equipas foi sempre focado nas características mais importantes.

A estimativa inicial para o projeto era incorreto, mas esta foi realizada fora do experimento Crystal Clear.

Crystal Clear exige um designer de chumbo experiente e competente, a fim de ser seguro. O Crystal Clear não define 'técnicas' (para gestão de configuração, testes etc), e a intenção é que o Designer de chumbo preenche esta lacuna.

As questões que requerem atenção durante a adopção

Os seguintes aspectos da metodologia precisam de atenção durante a adoção:

- Ela pode exigir papéis para mudar estabelecida

Crystal Clear dá às pessoas papéis que podem não corresponder as funções organizacionais estabelecidas.

No TRT Reino Unido, cada projeto tem um gerente de projeto que é responsável por entregar o projeto no prazo e dentro do orçamento. Em nosso projeto piloto, este foi o papel de Sam. Um gerente de projeto TRT! Reino Unido seria de esperar de possuir o cronograma do projeto. Mas sob Crystal Clear, a programação era uma atividade comum liderada pelo designer-chefe. Este foi um arranjo mais seguro, uma vez que trouxe as habilidades de todos os membros da equipas para suportar as

problema de programação. Mas criou uma situação em que Sam tinha a mesma responsabilidade que ele teria
normalmente, mas com potência reduzida para influenciar eventos 63.

- Ele desafia as práticas de documentação estabelecida

Crystal Clear espera requisitos iniciais na forma de uma declaração de missão, uma lista ator-objetivo, e alguns casos de uso. Estas não deverão ser suficientemente precisa para definir todos os aspectos do produto. Os detalhes são destinados a emergir por meio da interação entre representantes das organizações de desenvolvedores e clientes.

Crystal Clear também coloca grande dependência de conhecimento tácito sobre documentos escritos. Mas o conhecimento tácito está disponível apenas para outros membros da equipas, quando as pessoas estão simultaneamente presentes. Segue-se que, a fim de alcançar a máxima eficiência, os membros da equipas devem trabalhar juntos continuamente no mesmo projeto.

- Ela exige um contrato amplo de escopo

Crystal Clear espera que os requisitos detalhados para emergir como o projeto avança, mas é omisso quanto à questão de como isso poderia caber em um quadro contratual 64. Uma abordagem seria a adotar a prática DSDM de baselining os requisitos a um nível elevado. DSDM tem sido usado com sucesso em conjunto com o CMMI, por isso é pouco provável que esta abordagem criaria um problema cumprimento CMMI ou ISO.

Recomendações para trabalhos futuros

Este projeto-piloto exercido muitos aspectos da metodologia, mas foi limitado por seu pequeno tamanho. Recomendamos um julgamento em um projeto maior, com mais de 1_ desenvolvedores, e mais do que um único lançamento. Um teste maior ainda seria necessário no âmbito Crystal Clear, que não tenham mais de oito membros da equipas.

Crystal Clear é projetado para acomodar práticas extraídas XP, como o teste-primeiro projeto e programação em pares. Nós recomendamos que estas práticas ser incorporados em um julgamento futuro.

Um estudo sobre o ajuste entre Crystal Clear e nossa metodologia interna padrão está prevista para o final de 2004. Se possível, também gostaria de concluir o trabalho sobre o ajuste entre Crystal Clear e o CMMI.

Comentários da equipa

Os comentários a seguir são inéditos, exceto aonde indicado.

63 Eu discutir as opções que Sam possa ter no Comentário depois.

64 Isso é discutido no capítulo, *Questionou.*

Os comentários de Liam (utilizador)

11 º Dezembro de 2003:

Eu nunca trabalhei em um projeto em que os membros da equipas têm uma imagem tão clara do estado do progresso do projeto. As 'sessões de exibição e gerado 'regulares to- fazer' listas / erro de ser publicado em um quadro branco, cria um único ponto de acesso que todos os membros da equipas podem compartilhar. O plano de projeto e listas de tarefas associadas a ser postado-up também ajuda.

A única desvantagem que eu experimentei é o alto nível de conversa / discussão que é acontece em torno da placa principal, perturbando pensamento quando um não está envolvido em uma discussão equipas. Uma solução ideal seria ter o espaço para mover bordo e discussão de distância de membros da equipas não-envolvidos, durante essas reuniões.

3 rd Fev 2004:

A maneira pela qual os papéis metodologia Agile / cristalina tiveram que ser modificados de acordo com os procedimentos da empresa existentes, levou a um grau de reatividade no processo de avaliação. Por exemplo, o senior designer / coordenador (Stephen), apareceu a ser questionado sobre a gestão dos custos do projeto por parte do patrocinador, quando na verdade apenas o patrocinador ('Sam') tinha pleno acesso às informações de custo. Se a abordagem avaliada tinha sido totalmente integrado com os procedimentos da empresa estas questões não teria surgido. No geral, tomando parte foi uma experiência agradável, e que eu ficaria feliz em repetir.

Os comentários de Desmond (designer)

Focus & Eficiência

Como um designer / programador, eu costumo me senti muito "focado" e "energizado". Durante cada incremento eu senti que tinha uma ideia clara do que apresenta estaríamos adicionando, mais ou menos como eles devem trabalhar, quanto tempo eles devem tomar, e quais eram os mais importantes. Eu me senti em geral "sob controle".

O curto comprimento dos incrementos e a ideia clara de uma meta no final também ajudou a manter o foco no que é essencial. Cada incremento curta tinha um objetivo claro: um conjunto de funcionalidades a serem implementadas e uma demonstração de que para os utilizadores. O sentimento geral era de que estávamos a ser rápido e eficiente. Que estávamos focados claramente em recursos valiosos, e que fizemos progressos eficiente.

Em outros desenvolvimentos que se poderia ter a chance de "desaparecer" por alguns meses, trabalhando para um grande objetivo. Tendo contrastou isso com Crystal Clear, agora parece-me que Crystal Clear é muito mais eficiente: há sempre uma ligeira pressão para entregar, para se concentrar no que é essencial, que parece que agrega valor real. Ou melhor, que evita algumas das ineficiências da antiga abordagem.

Este sentimento energizado, focado é produzido por vários fatores:

- **Envolvimento no planificação de cada incremento**

- **Familiaridade com o conjunto de requisitos e as prioridades dos utilizadores (em termos de valor) para eles.**

- **Sensação de realização como cada incremento é completada, e dentro de incrementos, foi adicionado à medida que cada característica.**
- **Em menor medida, a ser libertado do "trabalho penoso" percebida de processos mais pesados. Na verdade, ele não se sentia como muito estava sendo descartado. Mas o knock-on efeitos de usar o processo mais leve definitivamente contribuiu para o bom sentimento.**

"Entregabilidade" e Controle

Em quase todos os momentos sobre o projeto parecia que tinha um produto de trabalho, com ideias claras sobre como proceder, e confiança de que algo útil seria entregue. Assim, não se sentir fora de controle, ou preocupado se nós realmente produzir algo no final.

Diferenças com métodos 'tradicionais'

Do meu ponto de vista - do ponto de vista de alguém que se deslocam de procedimentos de desenvolvimento "tradicionais" para Crystal, há dois princípios que parecem diferenciar Crystal Clear do antigo modo de trabalhar:

1) mínima sobrecarga necessário: fazer tudo da maneira "mais leve" possível, consistente com a obtenção dos resultados. Demorou um pouco para se sentir confortável com os processos e documentação mais leves de peso. Mas depois de um tempo tornou-se bastante fácil de manter, e muito libertador.

2) Ver feedback de utilizadores de uma forma positiva. Não ver descuidos ou alterações como problemas, mas como descobertas. Eu tinha que manter re-enfatizando isso para mim.

Comentários de Sam (patrocinador)

comunicações

Comunicações foram excelentes: a equipas principal foi co-localizado em uma única área. Um quadro impressão foi utilizado para gravar informação, que foi então realizada em um arquivo: isto provou ser uma poupança de tempo. No entanto, teria sido bom ter mais espaço para postar as informações para a equipas para ver.

Sessões e workshops de reflexão foram realizadas a cada mês e, certamente, manteve toda a equipas atualizado com o progresso demonstrável do projeto. No entanto, o número de iterações e projecções aumento de 3 a 5, e esta era excessiva.

Planificação / Gestão de Projetos

Do ponto de vista de um gerente de projeto, eu senti um pouco fora de controle por duas razões:

- **A gestão diária foi delegada ao Designer de chumbo,**
- **Eu não estava familiarizado com a metodologia Crystal Clear.**

Embora não faça parte da Crystal Clear, a proposta do projeto prometeu que o escopo do trabalho pode ser variada para manter dentro do orçamento original. Este não foi alcançado na prática e o projeto teve que ser reduzida em um ponto antes da conclusão integral.

Para projetos futuros, o papel do gerente de projeto precisa ser cuidadosamente pensado. O Crystal Clear não se encaixam bem com os procedimentos TRT existente no Reino Unido e por isso será necessário algumas mudanças.

Pessoas

A equipas trabalhou bem em conjunto, embora este projeto não era típico, em que a maior parte do desenvolvimento foi realizada por uma pessoa. O conceito de ter pouca documentação não funcionou bem nos estágios iniciais, porque os membros da equipas estavam de férias ou comprometido com outros trabalhos. É um crédito para a equipas que o projeto foi tão bem sucedida, apesar de isso.

entregas

O software foi entregue em uma condição de trabalho satisfatório. O documento de requisitos, manual do utilizador e manual de manutenção estão todos em fase de projeto, porque eles foram produzidos no final do projeto e do contingenciamento projeto impediu-os de serem revistos e emitido. Se os documentos foram produzidos anteriormente, não teria havido tempo para analisar e aprová-los. Eles também teria sido útil para orientar o trabalho de desenvolvimento. Na prática, uma auditoria ISO 9001 teria causado alguns problemas para o projeto.

Como o projeto foi encurtada, o software ainda não está pronto para ser lançado para os clientes, embora possa ser utilizado para demonstrações.

resumo

Para os projetos que podem ser claramente especificadas no início, Crystal Clear não parece oferecer vantagens significativas sobre um *bem gerida* projeto leve que segue processos padrão TRT Reino Unido. No entanto, aceita-se que para projetos como este, com uma grande componente Interfaces Homem-Computador, aonde os requisitos são difíceis de expressar claramente, Crystal Clear oferece algumas vantagens.

O uso de documentos em papel, quadros de impressão e de trabalho co-instalados podem não ser viável para uso generalizado e uma melhor abordagem seria a utilização de documentos eletrônicos como é prática corrente no TRT Reino Unido. Os problemas com os membros da equipas estar em licença em momentos críticos teria sido reduzido através da elaboração anteriormente documentação, mais detalhada. No entanto as sessões e workshops de reflexão eram muito importantes e devem ser mantidos como reuniões face a face.

[Fim do relatório]

O Relatório do Auditor

[CamCal era um experimento processo. O auditor analisou o que *seria* precisa ser feito para que um projeto Crystal Clear para passar um ISO 9001 de auditoria. O texto do auditor segue.]

Escopo

Esta auditoria é complementar ao Plano de Auditoria do Sistema de Gestão e foi realizada a pedido do 'Designer de chumbo' sobre os elementos CamCal do projeto ZYX.

Crystal Clear (CC) é uma metodologia de desenvolvimento de software que está sendo estabelecido para permitir perto de trabalho, colaboração de uma equipas de software de pequeno porte. O método é considerada '-peso leve' quando comparados a um desenvolvimento completo missão-crítica ou Safety-crítico (por exemplo, MIL STD 498, ESA PSS-05-0, etc.) e segue um ciclo de vida incrementais / evolutiva híbrido. Este tipo de abordagem é descrita como 'Rapid Application Development' (RAD) no Guia do TickIT.

CC exige uma estreita interação entre o cliente e os desenvolvedores e define uma série de papéis distintos.

CamCal havia adotado uma versão modificada do método CC como uma experiência consciente no desenvolvimento de software 'leve'.

O objectivo desta auditoria foi avaliar a metodologia de projeto CC aplicada ao CamCal contra os requisitos da ISO9001: 2000 e o Guia TickIT associado. Inevitavelmente, a implementação específica aplicada à CamCal influenciado a abordagem e os resultados da auditoria.

Duas listas de verificação foram preparadas a partir do Guia do TickIT. A primeira foi contra Parte D 'Orientação para Contas eo segundo foi contra Parte E 'Requisitos de Qualidade de Software Management System - Padrões Perspectiva'.

descobertas

Ambos ISO 9001: 2000 e TickIT são definidos no âmbito de um SGQ completo. Por conseguinte, como CamCal (ou qualquer outro projeto interno que segue a metodologia CC) irá operar dentro do TRT-Reino Unido, QMS elementos apenas o seguinte específico do projeto de norma ISO 9001: 2000 foram examinados:

Planificação da realização do produto

ISO / TickIT exigem planificação dos processos de realização do produto. Enquanto CamCal tem um plano geral do projeto 65 (que inclui os requisitos de qualidade) e Programação, não há nenhum plano detalhado do projeto para o documento desenvolvimento CamCal mas Proposta CamCal-01-003 foi referenciado a partir do Plano do Projeto.

documento de proposta CamCal-01-003 haviam sido emitidos e cobriu o planificação inicial de CamCal. Isto dá uma breve descrição do método e um ciclo de vida iteração propôs três. Uma das principais características do CC é que cada iteração planejada (havia cinco planejado mais tarde) inclui uma 'Reunião de Planificação', que permite que o projeto seja dinamicamente re-planejado. Uma iteração consiste em três etapas obrigatórias: planificação; design, código, integrar; e oficina de reflexão. Além disso uma iteração pode ser seguido por uma visualização com a presença do utilizador e, possivelmente, pelo patrocinador.

A fim de ser totalmente compatível com esta cláusula de ISO9001 **SGQ precisaria modificação para fornecer detalhes das práticas de trabalho associados com CC para evitar a necessidade de reproduzir esse detalhe no PP para cada projeto RAD. Um plano de projeto dedicado, então, precisa fazer referência ao QMS ou modificar as práticas, se necessário. O Plano de Projeto também precisa incluir ou fazer referência os seguintes (conforme apropriado): os requisitos de qualidade do cliente; Riscos ou processo de gestão de riscos; Métodos, ferramentas, padrões, etc .; requisitos estatutários e regulamentares; Entregas, incluindo documentos; Outros documentos; Verificação e validação abordagem; Aprovações, autorizações, aceitação; Sair da cama; prototipagem; Liberação.**

CC especifica uma 'Lista de Riscos', que precisa ser mantida. Recomenda-se que o processo normal de gestão de risco TRT-UK é seguido.

Determinação dos requisitos relacionados com o produto

ISO / TickIT requer determinação do requisitos relacionados com o produto. CamCal tem uma missão Patrocinador e uma evolução requisito Documento (RD) CamCal-02-003 tal como exigido por CC que continha uma lista Ator-meta e um número de casos de uso. A Missão tem numeradas e priorizados objetivos, o RD tem vindo a evoluir ao longo do desenvolvimento.

Estes documentos são capazes de satisfazer os requisitos / TickIT ISO. Foi sugerido que, no futuro, seria útil ter dois documentos distintos para distinguir entre os requisitos contratuais (declaração de missão além lista Ator-goal) e em evolução Requisitos CC documento que contêm casos de uso e detalhes de nível inferior e poderiam ser tratados como detalhe em vez de requisitos formais sob ISO 9001.

---

65 " plano de projeto" não se refere ao cronograma do projeto, mas com um plano para o *processos* usava..

Revisão dos requisitos relacionados ao produto

ISO / TickIT exigem revisão dos requisitos relacionados ao produto. CamCal envolveu análise e aprovação da declaração de missão e revisão em curso da RD. A revisão da RD foi realizado com o utilizador, em vez de Patrocinador e não houve questões formais (documento permaneceu em Issue 1 Projeto C no final, embora o projeto tinha sido encerrado prematuramente).

***De modo a ser completamente compatível com esta cláusula de ISO 9001,* RD devem ser submetidos a um projeto de revisão (com comentários mantidos em Visual SourceSafe) para cada iteração de projeto ou de preferência submetido a uma aprovado-se-emissão para cada iteração. Isso proporcionaria visibilidade das últimas exigências estáveis para cada iteração e os registros obrigatórios de avaliação exigidos pela ISO 9001.**

Se os requisitos contratuais foram tratados separadamente de detalhe, como sugerido acima, os requisitos contratuais só deve ser objecto de revisão na fase de proposta, conforme exigido pela ISO 9001.

comunicação com o cliente

ISO / TickIT exigem Comunicação com o cliente a ter lugar. O envolvimento do cliente (normalmente Patrocínio e Utilizador) é central para CC através de 'visões' e 'Workshops reflexão' durante cada iteração.

Consequentemente, este aspecto é bem coberta. CamCal realizado estas reuniões e mantidos registros de quadro branco.

CC requer uma estreita cooperação com representantes do cliente por um longo período. . escolha cuidadosa dos membros da equipas podem reduzir o risco de co-operação pobres.

Design e desenvolvimento de planificação

ISO / TickIT exigem planificação das atividades de design e desenvolvimento. Em TRT-UK esse detalhe normalmente seria contido em (ou referenciado a partir) do Plano e cronograma do projeto. Para CC, a maioria das práticas e iterações propostas poderia ser nesses mesmos documentos. CamCal não tinha um plano detalhado do projeto para CamCal mas tinha um documento de proposta e fez manter o cronograma.

***A fim de ser totalmente compatível com esta cláusula de ISO9001,* os requisitos definidos em 7.1 precisaria ser seguido. A programação seria necessário para refletir o mais recente 'Reunião de Planificação' (como faz em CamCal).**

Em CC planos podem ser alterados na reunião de planificação associado a cada iteração. Isso deve ser reconhecido no Plano de Projeto, e o escopo das mudanças que possam ser feitas sem revisão do Plano de Projeto deve ser definido.

entradas de projeto e desenvolvimento

ISO / TickIT requerem uma definição das entradas de projeto e desenvolvimento. CamCal colocado essa informação para um documento requisito (RD) CamCal-02-003 tal como exigido

por CC. A RD incluía a maior parte dos temas exigidos pelo TickIT. Infelizmente, a RD não foi emitido formalmente.

Se os requisitos contratuais foram tratados separadamente de detalhe, como sugerido acima, essa exigência seria atingido.

Em comum com a cláusula 7.2.2, *Para ser completamente compatível com esta cláusula de ISO 9001,* RD devem ser submetidos a um projeto de revisão (com comentários mantidos em VSS) para cada iteração de projeto ou de preferência submetido a uma aprovado-se-emissão para cada iteração. Isso proporcionaria visibilidade das últimas exigências estáveis para cada iteração e os registros obrigatórios de avaliação exigidos pela ISO 9001.

Design e desenvolvimento de saídas

ISO / TickIT exigem atributos relacionados com as saídas do projeto e desenvolvimento. CamCal não emitir todas as saídas definidas seguintes rescisão antecipada do projeto. Código da unidade passou por revisão por pares e verificar através da suíte Together. Aceitação O teste não foi realizado porque projeto foi reduzido.

***A fim de ser totalmente compatível com esta cláusula de ISO9001* cada projeto seria necessário para garantir a todas as ferramentas de software e COTS incorporado tinha sido, ou estavam a ser avaliada. código executável que precisa ser verificada durante o desenvolvimento e cada versão precisaria ser adequadamente definido e validado.**

De projeto e desenvolvimento

ISO / TickIT exigem revisão do projeto e desenvolvimento. 'visões' especifica CC e 'Workshops reflexão'. CamCal realizado estes para fora e mantiveram registros de quadro branco.

***De modo a ser completamente compatível com esta cláusula de ISO 9001,* registros de quadro seria necessário para indicar que estava envolvido com as sessões e workshops. Além disso, o patrocinador e / ou utilizador deve estar presente em algum ou todos, a fim de assegurar a independência revisor. As acções devem ser claramente identificados para permitir rastreamento no próximo workshop. Os registros do quadro branco que precisam ser mantidos para os registros obrigatórios de avaliação exigidos pela ISO 9001.**

verificação de design e desenvolvimento

ISO / TickIT exigir a verificação do projeto e desenvolvimento. CC especifica a verificação dos produtos de trabalho e CamCal fez realizar teste de unidade (embora há registros foram mantidos) e demonstração / revisão pelo utilizador durante a 'visões'. Falhas também não foram formalmente registadas durante o teste.

***De modo a ser completamente compatível com esta cláusula de ISO 9001,* registos de testes realizados (incluindo casos de teste / stimulii) e os resultados obtidos teria de ser levado para o**

registros obrigatórios exigidos 66. Revisão do desempenho em relação aos requisitos atuais deve ser realizado em estágios planejados.

validação de projeto e desenvolvimento

ISO / TickIT exigir a verificação end-to-end do projeto e desenvolvimento. CC especifica casos de teste, registros de teste e relatórios de defeitos, CamCal não produziu estes devido a rescisão antecipada.

***De modo a ser completamente compatível com esta cláusula de ISO 9001,* registos de testes realizados (incluindo casos de teste / stimulii) e dos resultados obtidos têm de ser tomadas para os registos obrigatórios exigidos. Alguma forma de rastreamento seria necessário que esses requisitos não foram testados ou demonstraram 67.**

Controle de alterações de projeto e desenvolvimento

ISO / TickIT requer controle de mudanças durante o projeto e desenvolvimento. mudança dinâmica é central para CC, que controla as alterações em dois níveis; mudanças para a missão ou exigências seriam feitas nos documentos distribuíveis; mudanças no projeto são menos formalmente controlada por meio de oficinas e reuniões de planificação. CamCal mantido 'acções Lists' e conclusão gravado em visões. testes de regressão foi utilizado para verificar os efeitos das mudanças mas os registros formais não foram mantidas.

***Para ser totalmente reclamação com esta cláusula da ISO 9001,* registros de quadro seria necessário para indicar que estava envolvido com as sessões e workshops, a fim de fornecer os registros obrigatórios exigidos para análise das mudanças. Como acima, os registos dos testes de regressão realizadas (incluindo casos de teste / stimulii) e os resultados obtidos também devem ser tomadas.**

Conclusões e Recomendações

Projeto CamCal adotado método rápido desenvolvimento do 'Crystal Clear' como uma experimentação e demonstração dentro TRT-UK de como poderia ser utilizado o processo. O cliente para o projeto foi interna (daí proteger todos os clientes externos 'reais' do experimento), mas o projeto foi, infelizmente, terminou cedo. No entanto, devido à natureza do método de CC, a rescisão antecipada não proibir um produto de software útil.

Esta auditoria avalia a metodologia CC contra a ISO 9001: 2000 eo Guia TickIT associado, usando CamCal como estudo de caso. Inevitavelmente, os processos detalhados seguido pelo projeto teve alguma influência sobre as observações e conclusões.

CC é geralmente compatível com a norma ISO 9001, especialmente considerando que o Guia do TickIT reconhece e acomoda 'Rapid Application Development' (RAD).

---

66 Aqui é aonde testes de regressão unit- e aceitação automatizados vir a calhar - Alistair

67 Este refere-se a usabilidade e outros requisitos que não têm um teste de função.

O cumprimento integral exigiria algumas disciplinas extra / registros, conforme descrito em 'achados'.

É **recomendado que:**

- **O SGQ TRT-UK é aprimorado para fornecer um quadro genérico para a metodologia CC (de preferência chamado algo como 'Método rápido desenvolvimento', como CC é algo de uma marca). As melhorias devem incluir os requisitos obrigatórios ISO.**

- **Cada projeto CC ainda deve criar um plano de projeto para adotar ou modificar a abordagem genérica.**

- **Além disso in-house desenvolvimentos são realizados para CC, a fim de compreender plenamente a aplicação e desenvolver os novos procedimentos.**

Uma vez estabelecida, clientes externos são dadas a escolha da metodologia de desenvolvimento, incluindo informações sobre os processos, os benefícios potenciais e as suas obrigações.

Reflexão sobre o campo e relatórios de auditoria

Primeiro de tudo, gostaria de agradecer a ambos Stephen e Thales Pesquisa e Tecnologia para o relatório de campo excelente, completo, pois é com fotos e entrevistas, e permissão para publicá-lo, mesmo em forma Extraído. Falando como alguém que tenha lido um monte de relatos de experiência, acho que é excepcionalmente clara e informativa, com detalhes úteis e franqueza.

Deixei tudo de questionamento dos participantes e se preocupar com Crystal Clear no lugar porque eu suspeito que estes serão reações comuns em muitas organizações, e é útil para que você possa vê-los colocado para fora na frente de você.

Nesta seção, considero duas de suas preocupações e discutir como lidar com eles em diferentes contextos. Com sorte, isso vai ajudar a Thales no futuro, e você também.

Realocação de Energia

O gerente de projeto e do patrocinador, Sam, mencionou várias vezes que ele se sentia um pouco sem poder, não tendo controle sobre as estimativas de tempo de tarefas e gestão quotidiana. Responsabilidade sem autoridade
é uma coisa preocupante. Sam bastante razoável solicitar a revisão do papel do gerente de projeto em Crystal Clear.

Ao discutir isso com Stephen, descobrimos uma série de questões e estratégias em torno deste assunto.

Em primeiro lugar, o poder realmente se realocados, em uma das duas direções. O patrocinador tem a responsabilidade de fornecer prioridades claras e resultados preferidos quando trocas de direção precisa ser feita. Os desenvolvedores têm a responsabilidade de estimar quanto tempo as tarefas irá tomar. Juntos, eles têm a responsabilidade conjunta para citar todas as tarefas, e para gerar a melhor estratégia que irá produzir o melhor resultado para o projeto, dada a lista de tarefas, as estimativas de tarefas e prioridades do projeto.

Em algumas empresas, o gerente de projeto corre o show, nomeando as tarefas, as estimativas e as estratégias. Tendo sido nessa posição durante anos, posso dizer que esta é uma proposta perdedora. É quase impossível para o gerente de projeto para adivinhar corretamente todas as tarefas que precisam ser feitas e as suas estimativas. A partir do dia em que eu vi pela primeira vez Jens Coldewey executar uma sessão de planificação conjunto, eu percebi o quão impossível.

Em outras empresas, os programadores executar o show, dizendo o patrocinador que eles vão fazer e quando pode ser feito, sem se importar com o que as prioridades do negócio estejam. Isso também é incorreto.

Em cada caso, é provável que alguém se sentir deslocada pela redistribuição de poder no Crystal Clear (e outros métodos ágeis). A chave para o sucesso é o reconhecimento de que todos estão juntos nessa, o projeto é uma atividade conjunta, e ambos os grupos precisam fornecer as informações que possuem, a fim de obter o melhor resultado.

Em segundo lugar, o patrocinador ainda não está sem energia elétrica.

Suponha que a sessão de planificação produz uma linha do tempo que o patrocinador considera inaceitável. Ele tem três recursos:

- **Remover alguns dos itens a serem entregues ou prolongar o período permitido.**
- **Ser mais criativo sobre o plano: descarregar os membros da equipas, fazer mais em paralelo, ou comprar pacotes que realizam parte do trabalho. Muito pode ser feito com um brainstorming criativo.**

- **Reduzir todas as estimativas de tempo por uma porcentagem. Sim, esta é uma medida drástica, mas ainda é uma alternativa. Imagine que o patrocinador diz: "Tudo muito bem, mas não podemos viver com essa programação e não podemos cortar mais qualquer escopo. Vou te dar 80% do tempo listado em cada cartão, para que possamos fazer a data final ". Neste ponto, todo mundo já viu as cartas, para que saibam as estimativas originais. A equipas pode acompanhar a ambas as estimativas originais e reduzidos e relatar contra ambos em seus mapas de status. Visibilidade será mantido alta. Eu suspeito que se o patrocinador sente que ele deve recorrer a esta alternativa, então não é uma metodologia de desenvolvimento diferente, que irá produzir um melhor resultado de qualquer maneira. Sugiro também que a equipas manter brainstorming para uma estratégia melhor.**

"Ela exige um contrato amplo de escopo"

O relatório de campo conclui, "Crystal Clear espera que os requisitos detalhados para emergir como o projeto avança, mas é omisso quanto à questão de como isso poderia caber em um quadro contratual. Uma abordagem seria a adotar a prática DSDM de baselining os requisitos em um alto nível."

CamCal foi baseado em um rascunho de Crystal Clear. Espero que eu consegui nesta versão final para mostrar como ele pode trabalhar em um quadro contratual.

Crystal Clear não *exigir* os requisitos detalhados para emergir como o projeto avança, embora *permitem* -los para. Você não são de todo obrigado a mudar os requisitos no início de uma iteração. Se você tiver requisitos detalhados no início que não têm permissão para mudar, então simplesmente não transferi-los!

Pode ser que você se comprometeram a um escopo, preço e tempo sem ter tido tempo para investigar adequadamente os requisitos e arquiteturas plausíveis (isto é distressingly normal em nossa indústria). Você descobre a meio do projeto que a oferta original era simplesmente errado. Você tem várias opções, nenhuma delas muito:

- **Volte para os patrocinadores / compradores e renegociar. Esta não é uma questão para o Crystal Clear ou qualquer outra metodologia para lidar com eles. Isto tem tudo a ver com o quão bom relações do seu executivo estão com os compradores e que eles podem mudar. Eles podem ser capazes de renascer o projeto, ou eles não podem.**
- **priorizar cuidadosamente o seu trabalho para entregar o valor máximo de negócios no tempo disponível, de modo que as características deixadas de fora no prazo são óbvios para todos como sendo os menos importantes. Tendo entregue o que é claramente**

a funcionalidade de negócio mais valioso é a melhor moeda de troca que você pode dar aos seus executivos para trabalhar com 68.

- **Ser muito criativo na sua geração de estratégia e encontrar uma maneira de entregar o lance.**
- **Prepare-se, quer a trabalhar horas extras ou mudar de emprego. Mesmo o melhor metodologia ágil não pode mudar suas regras sociais (mas pode torná-lo um aluguer mais atraente).**

* * *

I deve completar esta reflexão com uma nota final sobre "Sam", o patrocinador. Stephen escreve: ".. No último workshop de reflexão, o patrocinador também disse que as vantagens superavam em muito as desvantagens. Ele sugeriu que tente isso de novo, em um projeto maior, desde que o ajuste com os nossos procedimentos existentes podem ser resolvidos."

Relevantes para os testes para o projeto metodologia, o projeto entregue um resultado adequado para o patrocinador, e as pessoas (incluindo o patrocinador) estão dispostos a usá-lo novamente.

68 Veja a discussão em *Victory cedo (* p. 67), *Queime gráficos (* p. 118) e *Interaction Design Essencial (* p. 105).

Capítulo 9 destilada (A versão curta)

No final, é hora de rolar tudo de volta novamente: O que é o núcleo do Crystal Clear, e quais são os add-on práticas que começam a equipas mais para dentro da zona de segurança? Este capítulo é muito curto

.

Crystal Clear é uma forma altamente otimizado para utilizar uma equipas pequena, colocated, priorizando para *segurança* na entrega de um resultado satisfatório, *eficiência* em desenvolvimento, e *habitabilidade*
das convenções de trabalho.

A breve descrição do Crystal Clear para nível 3 praticantes é apenas isso:

O designer-chefe e dois para sete outros desenvolvedores em
uma grande sala ou salas adjacentes,

usando radiadores de informação, tais como quadros e cartazes, tendo fácil
acesso a utilizadores experientes, distrações mantidos longe,

entregar em execução, testado código, utilizável para os
utilizadores a cada mês ou dois (pelo trimestral pior),

refletindo e ajustar suas convenções de trabalho periodicamente.

As pessoas definidas no lugar as propriedades de segurança abaixo usando as técnicas que eles sentem apropriado. As três primeiras propriedades são necessárias no Crystal Clear; os próximos quatro obter a equipa mais para a zona de segurança.

1. **Entrega Frequent**
2. **Melhoria reflexivo**
3. **Comunicação osmótica**
4. **Segurança pessoal**
5. **Foco**
6. **Fácil acesso a utilizadores experientes**
7. **Um ambiente técnico com testes automatizados,**
   Configuração Gestão e Integração Frequent

Todas as outras páginas deste livro só expandir nesta página.

Verestatísticaspublicaçãode publicaçãoEstatísticas Ver as