Revista Visão Ágil

Seu canal de conhecimento sobre processos ágeis

Ano I - Edição 01 – Julho de 2007 - www.visaoagil.com

Liderança

Nesta edição, trazemos artigos especiais sobre esse tema.

Veja nesta edição:

O Pink e o Cérebro

Crônica sobre a difícil arte da simplicidade em projetos de software através da análise desses dois perfis.

Contrato de Escopo Negociável

Veja nesse artigo, o que é , como funciona, e como aplicar esse modelo em sua empresa.

Planejando seu projeto com Extreme Programming – Parte I

Aprenda como funciona a estrutura base das práticas de Extreme Programming, e veja como iniciar um planejamento de projetos seguindo suas idéias.

Mostra-me tua equipe, e eu te direi quem és!

Artigo que lhe levará a uma auto-avaliação sobre suas atitudes como líder ou participante de um projetos.

Scrum e as armadilhas das reuniões diárias!

Entenda como funciona e aprenda como evitar as armadilhas das reuniões diárias em seus projetos.

As Cinco Doenças do Gerenciamento de Projetos

Veja nesse artigo a tradução para o português de um clássico no mundo dos projetos.

Nosso “Selected Backlog”

Editorial

Especial sobre a primeira edição e como funcionará as próximas.

Alexandre Magno Figueiredo

Revista Visão Ágil

Equipe Visão Ágil
Diretor Editorial: Manoel Pimentel Medeiros
Revisão: Manuella Bulcão Medeiros
Editor Adjunto: Alexandre Magno Figueiredo

Atendimento ao leitor
e-mail: manoelp@gmail.com
site: www.visaoagil.com

Edições Anteriores
Qualquer edição anterior pode ser baixada gratuitamente no site **www.visaoagil.com**

Publicidade
Se você está interessado em fazer alguma ação de marketing em parceria da Revista Visão Ágil, entre contato conosco através do e-mail **manoelp@gmail.com**, e teremos o maior prazer em trabalharmos juntos.

Visão Ágil: agilidade do início ao fim

Uma das primeiras perguntas que nos fizemos ao termos a idéia da Visão Ágil foi: como elaborarmos uma revista de forma ágil? De uma coisa sabíamos, ela não poderia ser elaborada da mesma maneira que as outras revistas, precisávamos de algo mais...ágil! Bom, fico feliz em comunicar a todos que a Revista Visão Ágil é agilidade pura, e utiliza o framework **Scrum** em todo o seu processo de criação e desenvolvimento. Optamos por fazermos esta **edição 01** da revista com artigos já conhecidos de nomes consagrados da comunidade ágil brasileira, seu maior propósito é a divulgação da marca e do processo de elaboração que a mesma adotará a partir da segunda edição...portanto, apertem os cintos e sejam bem vindos à agilidade do início ao fim!

Como funcionará a Visão Ágil?

A Revista Visão Ágil possui um **Product Owner**, e ele será o elo entre todos vocês **stakeholders** aos nossos articulistas (desenvolvedores) e ao nosso **Scrum Master** (Editor). No nosso site haverá uma área para sugestões de artigos e temas, essa área será gerenciada pelo nosso Product Owner que, através das solicitações de vocês, montará o **Product Backlog** da Visão Ágil, definindo o business value de cada sugestão de artigo de acordo com a quantidade de solicitações para o mesmo tema. Em cada data agendada para o início do Sprint de desenvolvimento da próxima edição da revista, o time realizará o **Planning Meeting** e, de acordo com o Product Backlog, escolhará o que "entrará" na próxima edição, ou seja, o que tem realmente valor para vocês, nossos leitores. É em cima do Product Backlog selecionado que nosso time trabalhará, buscando os melhores articulistas para cada assunto selecionado e iniciando a Sprint. Paralelamente a isso nosso Product Owner continuará recebendo as solicitações de vocês e atualizando o Product Backlog para as edições seguintes.

Site da Revista Visão Ágil no ar
Ótima notícia para todos os agilistas, o site da revista Visão Ágil, já se encontra no ar.
Acesse www.visaoagil.com e veja que você pode participar com nossa equipe editorial, bem como ter acesso à outros conteúdos referentes ao universo ágil.

Curso de Scrum pela AxMagno
O nosso amigo Alexandre Magno, está preparando uma novidade legal para nós agilistas, ele está preparando um curso prático em parceria com a empresa de treinamento Caellum de São Paulo.
Informações em:
www.axmagno.com

Heptagon fecha parceiria com IVIS
A Empresa Heptagon, que têm desenvolvido um trabalho pioneiro em **FDD**(Feature Driven Development), fechou um acordo comercial com a empresa Americana IVIS, produtora do XProcess, ferramenta que atende a criação de templates para cobrir quaquer processo.
Maiores informações em:
www.heptagon.com.br

Enquetes sobre adoção dos processos ágeis:
Atenção membros do **GUFDD** (Grupo de Usuários de FDD) e do **XPNorte**(Grupo de Usuário XP do Região Norte do Brasil), participem das enquetes postadas em cada grupo, pois são temas importantes que podem mostrar o cenário do desenvolvimento ágil no Brasil.
O site do **XPNorte** é:
http://br.groups.yahoo.com/group/xpnorte
O site do **GUFDD** é:
http://br.groups.yahoo.com/group/gufdd/

Reuniões Mensais do Grupo XPRio

Sempre às 19 horas, no Auditório do Centro de Informática e Telecomunicações do SENAC Rio (Cinelândia - RJ).
A entrada é gratuita e não é necessário fazer inscrição. A data e o tema do mês são sempre divulgados na página do XPRio em **http://xprio.blogspot.com/.**

O Pink e o Cérebro

Crônica sobre a difícil arte da simplicidade em projetos de software

Manoel Pimentel Medeiros

Pink e Cérebro, esses dois ratinhos(opa, camundongos de laboratório), talvez sejam um dos recentes desenhos animados, mais interessantes em minha opinião, principalmente pelo antagonismo dos personagens principais, dessa forma, quero tentar usá-los nessa crônica para ilustrar o ponto de equilíbrio ideal entre soluções tecnológicas de topo e simplicidade.

Para isso, vamos analisar o perfil de cada um deles, começaremos pelo Cérebro, sujeito pensador, com alto nível intelectual, e uma incrível capacidade de planejar e arquitetar soluções altamente complexas para executar atividades simples, por exemplo, ele criaria um super mecanismo, com tubos movidos a pressão hidráulica, iniciada por um coletor de energia solar, armazenada em um dispositivo de armazenamento termo-nuclear só para criar um canudo para tomar suco de limão. Não que isso seja algo totalmente ruim, na verdade, o Cérebro só é capaz de criar esse tipo de coisa, devido a sua inteligência extremamente exagerada(igual a sua cabeça), porém trazendo para nossa realidade, ele representa um gerente ou arquiteto de software que cria um projeto altamente super-dimensionado e usando várias combinações de tecnologias e técnicas só para criar uma simples tela de cadastro acessado um banco de dados.

“não é porque um médico têm conhecimento para fazer uma cirurgia neurológica, que ele vai usar essas mesmas técnicas e materiais para suturar um pequeno corte no dedo.”

Já o Pink, é um cara do tipo bobalhão, meio pateta que acredita em tudo que o Cérebro lhe fala, porém, ele têm uma característica bem valiosa, sua tendência a simplificar as coisas, pois através de suas ações baseadas em sua intuição, ele consegue ter resultados muito mais eficazes que o Cérebro, dessa forma, ele representa quem prefere soluções mais simples e fáceis para criar a mesma tela de cadastro acessando um banco de dados, porém, pode ter uma grande limitação de visão quando for necessário resolver problemas que exigem mais complexidade. Na verdade, a grande sacada é que nem o Cérebro com seu intelecto superior, nem Pink com sua visão extremamente simplista dos problemas, quando sozinhos, conseguem chegar a algum lugar, eles só se dão bem, quando há uma parceria, ou seja um ponto de equilíbrio entre soluções inteligentes e simples. Esses dois perfis, são o exemplo claro de comportamentos muito comuns em um projeto de software, pois por muitas vezes, projetamos verdadeiras naves espaciais quando precisamos apenas inicialmente de um simples carro com motor 1.0, porém(sempre têm um porém), pode haver momentos que será preciso nosso carro ter um motor 1.8 turbinado, aí entra o verdadeiro dilema, pois se pensamos exageradamente simples, podemos pecar por falta de

visão ou sub- dimensionar uma solução, que também seria prejudicial ao sucesso de um projeto.

Então meu caro amigo, agora você deve estar se perguntando, “e agora quem poderá nos defender?”, e eu lhe digo “calma, calma, não criamos pânico..”, pois a solução para isso é ter **bom senso** em todas as fases de um ciclo de vida de um projeto, ou seja, tenha em mente o seguinte exemplo: não é porque um médico têm conhecimento para fazer uma cirurgia neurológica, que ele vai usar essas mesmas técnicas e materiais para suturar um pequeno corte no dedo.

Portanto, meu caro amigo, conheça as tecnologias, busque conhecimentos, explore suas possibilidades, mas sempre lembre que, um projeto é um empreendimento, e que todas as partes querem ter bons resultados com ele, por isso, sempre que possível, privilegie soluções simples, pois dessa forma, você estará diminuindo a complexidade e muitas vezes, amenizando o impacto negativo que um projeto complexo pode ter, e maximizando seu nível de produtividade, ou seja, use a tecnologia para o que ela realmente foi feita ajudar e não prejudicar você, pense nisso

Manoel Pimentel Medeiros
(manoelpimentel.blogspot.com) É Engenheiro de Software, atua em projetos ágeis pela **Heptagon**(SP), entre outras atividades, é diretor editorial da **Revista Visão Ágil**, líder do projeto **BoxSQL** , se dedica a coordenação do grupo de usuários de extreme programming da região Norte (XPNorte) e do **NUG-BR** (NetBeans Users Group - Brasil), além de ser colunista do portal e revista **Java Magazine**, possui também as certificações: Scrum Master, Project Management, Java e Delphi da Brainbench.

Planejando seu projeto com Extreme Programming – Parte I

Manoel Pimentel Medeiros

Talvez você já tenha lido vários textos sobre **XP (eXtreme Programming),** talvez também você até diga "legal, mas como eu uso isso?", portanto, esse artigo, almeja explicar de maneira simples e objetiva como você pode iniciar seu projeto usando as premissas fundamentais do desenvolvimento ágil.

Só para relembrar, XP, é um processo de desenvolvimento de software que é baseado em premissas ágeis como:

- Desenvolvimento incremental;

- Projetos preferencialmente com um escopo orientado a objetos;

- Escopo variável, ou seja, os requisitos são vagos e mudam com freqüência.

Entenda que o termo ágil, não significa que o código do projeto vai ser produzido como um passe de mágica da noite para o dia, na verdade, os processos ágeis como XP, se baseiam na idéia de prover a todas as partes de um projeto, o máximo de produtividade por cada dia de trabalho, gerando dessa forma um maior valor agregado e um alto nível de qualidade ao software desenvolvido.

Essa agilidade, conforme já falei em artigos anteriores, é assegurada por um conjunto de valores como **feedback**, **comunicação**, **simplicidade, respeito e coragem,** através também de uma gama de práticas como:

- O Cliente Presente,

- Estórias,

- Jogo do Planejamento,

- Programação em Par,

- Reuniões em Pé(rápidas),

- Desenvolvimento guiado por testes,

- Testes de Aceitação,

- Refactoring,

- Código Coletivo,

- Padrões de Código,

- Simplicidade de Design,

- Metáforas,

- Ritmo Sustentável,

- Integração Contínua,

- Releases Curtos

Como você pode ver o estudo sobre XP apesar de ser bem detalhado, é bem simples de compreender e aplicar, dessa forma, esse artigo terá como foco, a aplicação de alguns "**artefatos**" que servem como ferramentas fundamentais para o uso de XP.

Por isso, iremos inicialmente estudar nesse artigo as idéias básicas de uma estória e como são estruturados as iterações e os releases, mas nos próximos, iremos estudar o que é um dia perfeito de trabalho? Como funciona o sistema baseado em pontos? Como compor do cartão de estória? E por fim, como elaborar um roteiro para testes de aceitação.

O que é uma estória?

Apesar de que falaremos mais detalhadamente sobre esse tema nos próximos artigos, decidi colocar aqui uma breve introdução sobre o que são estórias, pois é crucial que você tenha esse entendimento básico, para conhecer o funcionamento de XP como um todo.

Estórias são as funcionalidades que o cliente espera receber na aplicação que será construída, que em outras abordagens seriam chamadas de requisitos, casos de usos ou pontos de função.

Quem escreve uma estória?

Normalmente, elas são registradas em cartão de estória, pelo próprio cliente, em uma linguagem simples e comum o suficiente para a compreensão de todos da equipe.

Quem usa uma estória?

Como você pode observar na **figura 01**, os maiores usuários das estórias, são os desenvolvedores, que o fazem a fim de guiar o que deve ser desenvolvimento na aplicação, porém, costuma-se afirmar que as estórias são um "**convite ao diálogo**" entre os desenvolvedores e o cliente, ou seja, não é necessário que as estórias sejam bem detalhadas, na verdade é recomendado que o texto contido nelas, seja extremamente simples e objetivo, e principalmente, deixe uma margem para estimular o diálogo entre a equipe e o cliente a fim de dirimir qualquer dúvida sobre a funcionalidade em questão.

É importante observar que é através dessas estórias que podemos estimar qual será o esforço necessário para implementar essa tal funcionalidade através de um sistema baseado em pontos(que veremos no próximo artigo).

Essas estórias, também fazem parte de um conjunto de outras estórias que serão agrupadas em alguma iteração.

E só para deixá-lo com bastante curiosidade, no próximo artigo, iremos estudar como compor um cartão de estória.

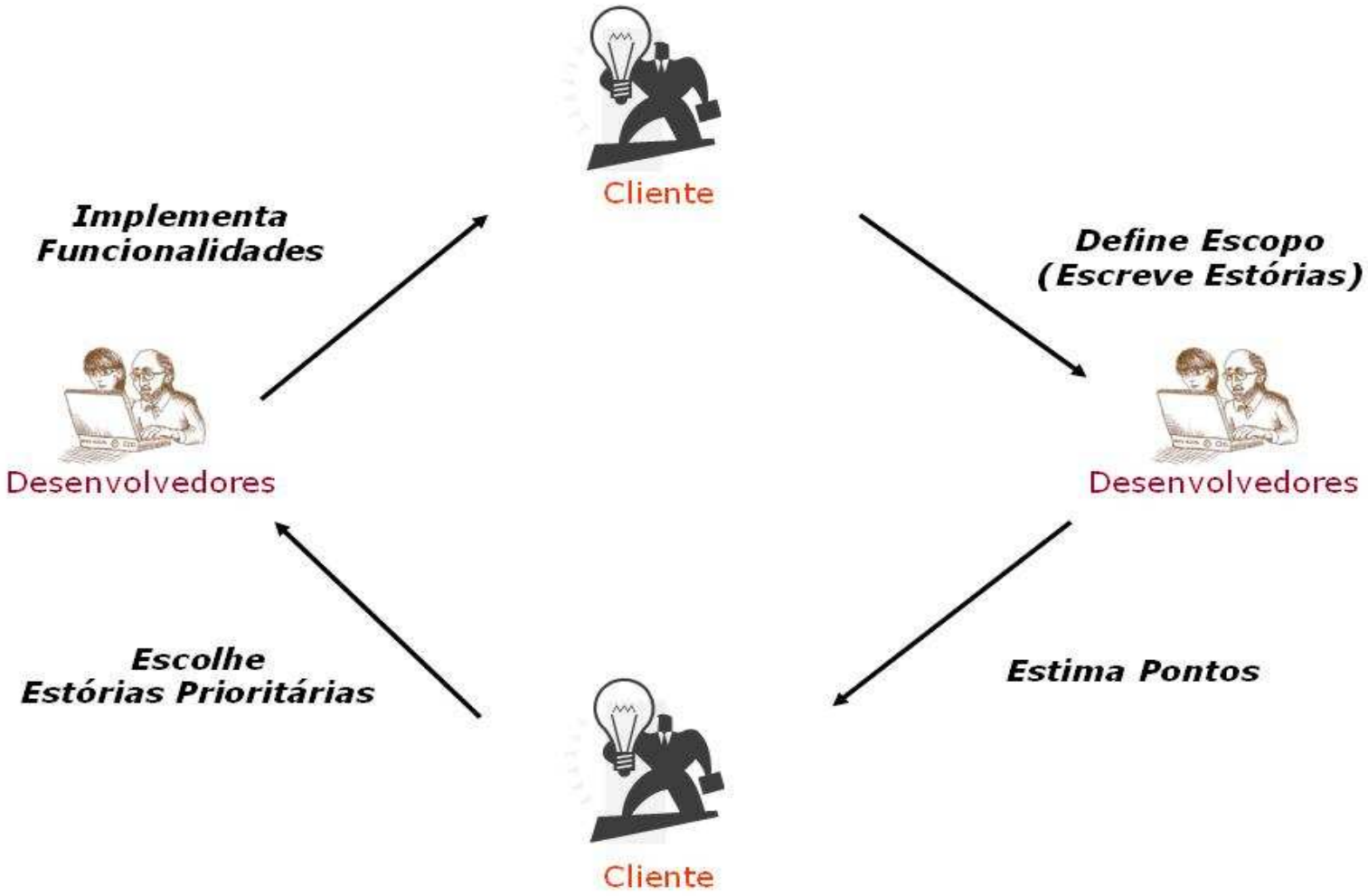

Figura 01 – Ciclo de vida de uma estória em projeto XP

O que é uma iteração?

Uma iteração de uma maneira bem simples, é o tempo disposto a fim de gerar a implementação de um determinado número de estórias, é importante lembrar que é na iteração que iremos aplicar o conceito de modelo em espiral que veremos mais adiante.

Quanto tempo dura uma iteração?

Sem parecer clichê, o tempo de duração é relativo, pois vai depender do tipo do projeto, tamanho da equipe, capacidade técnica da equipe ou necessidade do cliente.

Convenciona-se apenas que uma iteração deve ter o menor tempo possível, pois é através dela, que conseguiremos medir e corrigir o andamento de um projeto, por isso é comum encontrar iterações que variam de uma a três semanas, mais como eu disse anteriormente, você define isso de acordo com sua necessidade.

Como se inicia uma iteração?

Normalmente dedica-se um dia inteiro, ou boa parte dele, para uma reunião que é chamada de **Jogo de Planejamento,** onde, programadores, gerentes e clientes definem e estipulam quais as estórias serão priorizadas naquela iteração, como serão seus roteiros de testes, quais serão as combinações iniciais de pares, o que houve de errado na iteração anterior, e que pode ser melhorado naquela iteração atual.

Como se termina uma iteração?

Terminado ora bolas! (eheh brincadeira!) Basicamente ao término de uma iteração, dedica-se também um dia inteiro para a realização de testes de aceitação, verifica-se principalmente, quais estórias foram bem sucedidas e quais passarão para a próxima iteração como uma estória de **correção**.

Dica:

Para as equipes mais animadas e **PRODUTIVAS**, costuma-se fazer ao fim de cada iteração, um bom happy-hour para comemorar ou "bebemorar" o sucesso da mesma. Acredite, isso tem um impacto muito estimulante para toda a equipe, portanto se possível, **EXPERIMENTE**!

O que é um Small Release?

Primeiro é importante que você entenda muito bem, que um **release curto** é uma **pequena versão** do sistema, que comumente é funcional o suficiente para que o cliente possa começar a usá-lo a fim de fazer testes mais funcionais, iniciar os cadastros básicos, começar a sentir se o caminho que o projeto está indo, é o correto ou até mesmo liberar aquela **pequena versão** para o uso por parte dos usuários do sistema.

É importante, notar, que essa idéia possibilita o cliente, ver de maneira mais rápida, o retorno sobre seus investimentos naquele projeto, isso é importante, pois dessa forma, o cliente terá um controle maior sobre a evolução do mesmo e principalmente, poderá implantar o sistema sobre demanda, sem necessariamente, depender do fim do projeto inteiro, para ver como o sistema funciona e se comporta.

Estrutura baseada em releases e iterações

Ao contrário das metodologias de desenvolvimento tradicionais, que seguem idéias muitos próximas ao modelo em cascata, os processos ágeis como XP, usam um modelo **incremental**, que se fosse traduzido graficamente, seria semelhante a uma espiral.

Na verdade, o modelo espiral não representa o projeto como um todo, pois, ele é usado várias vezes em pequenas partes durante o mesmo, formando ciclos que se repetem até o final do projeto.

Nesses **ciclos** que chamamos de **iterações** vemos claramente que são miniaturas de projetos, onde aplicamos as fases presentes em um desenvolvimento mais tradicional, ou seja, seria como tivéssemos **vários pequenos projetos dentro um projeto só**.

Cada iteração terá um conjunto de estórias a serem **implementadas**, que passam por uma **análise**, por um **design**, e por **testes** na mesma iteração vigente.

Veja na **figura 02**, como é estruturando um projeto seguindo o modelo incremental, observe que em cada **iteração**, aplicamos o modelo espiral, repetindo as fases de **análise**, **design**, **implementação** e **testes**.

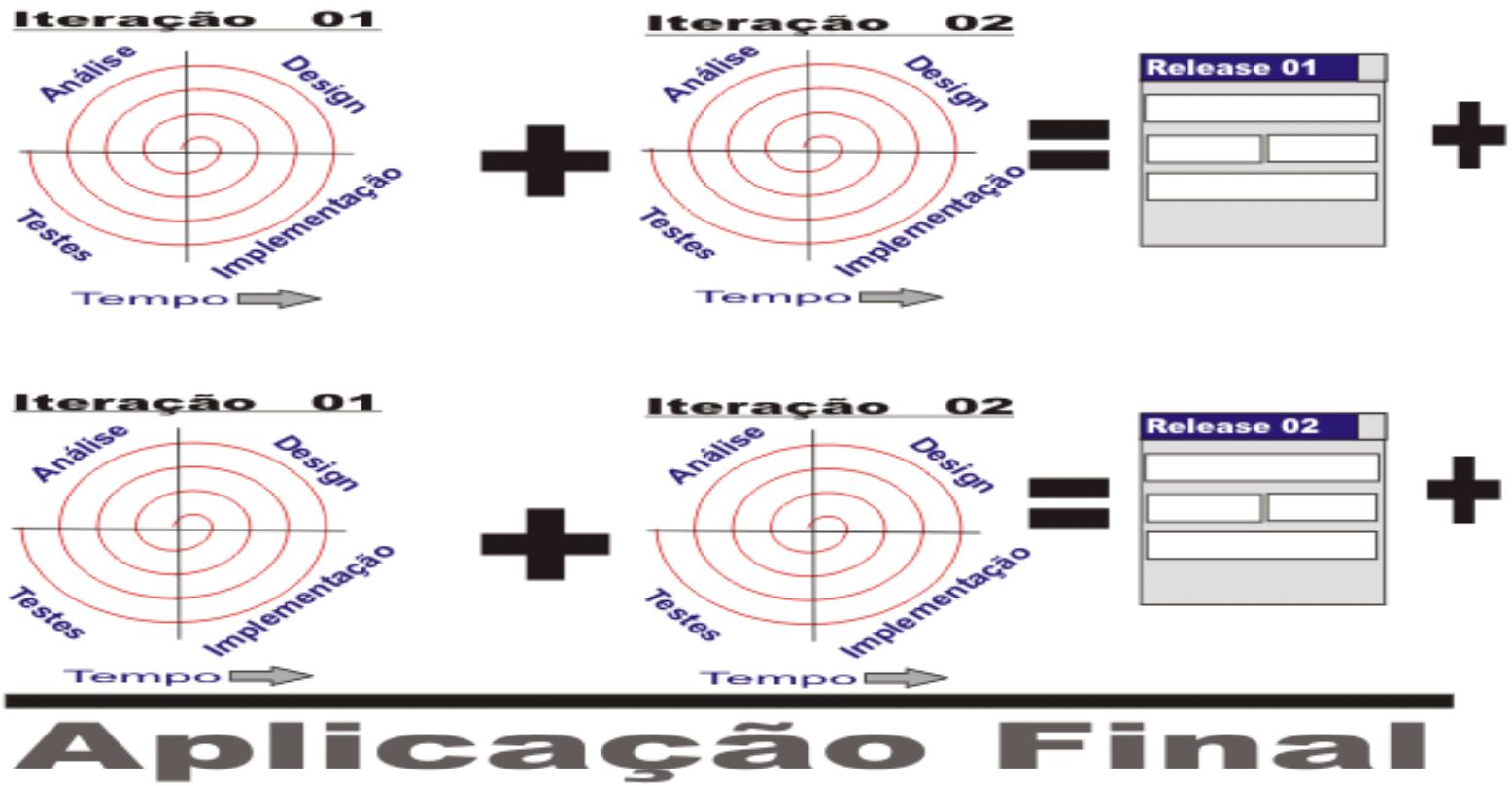

Figura 02 – Estrutura de um projeto com o modelo incremental

Esse modelo se apóia na premissa ágil de que por vários motivos o escopo do projeto muda no decorrer de sua execução, portanto, essa estrutura é capaz de absorver de uma forma mais eficaz essas possíveis alterações.

Isso evidencia o fato de que é mais simples alterar ou corrigir pequenos pedaços do projeto, do que fazê-los no software inteiro e somente no final do mesmo.

É muito importante que você entenda muito bem esse conceito, pois o mesmo é a base de todo o dinamismo proposto por XP. Para isso veja na **figura 03**, outro diagrama que mostra outro prisma de uma estrutura de projeto usando XP, observe que teremos a figura dos **releases**, e que estes por sua vez possuem as **iterações** que também possuem o conjunto de **estórias** a serem implementadas.

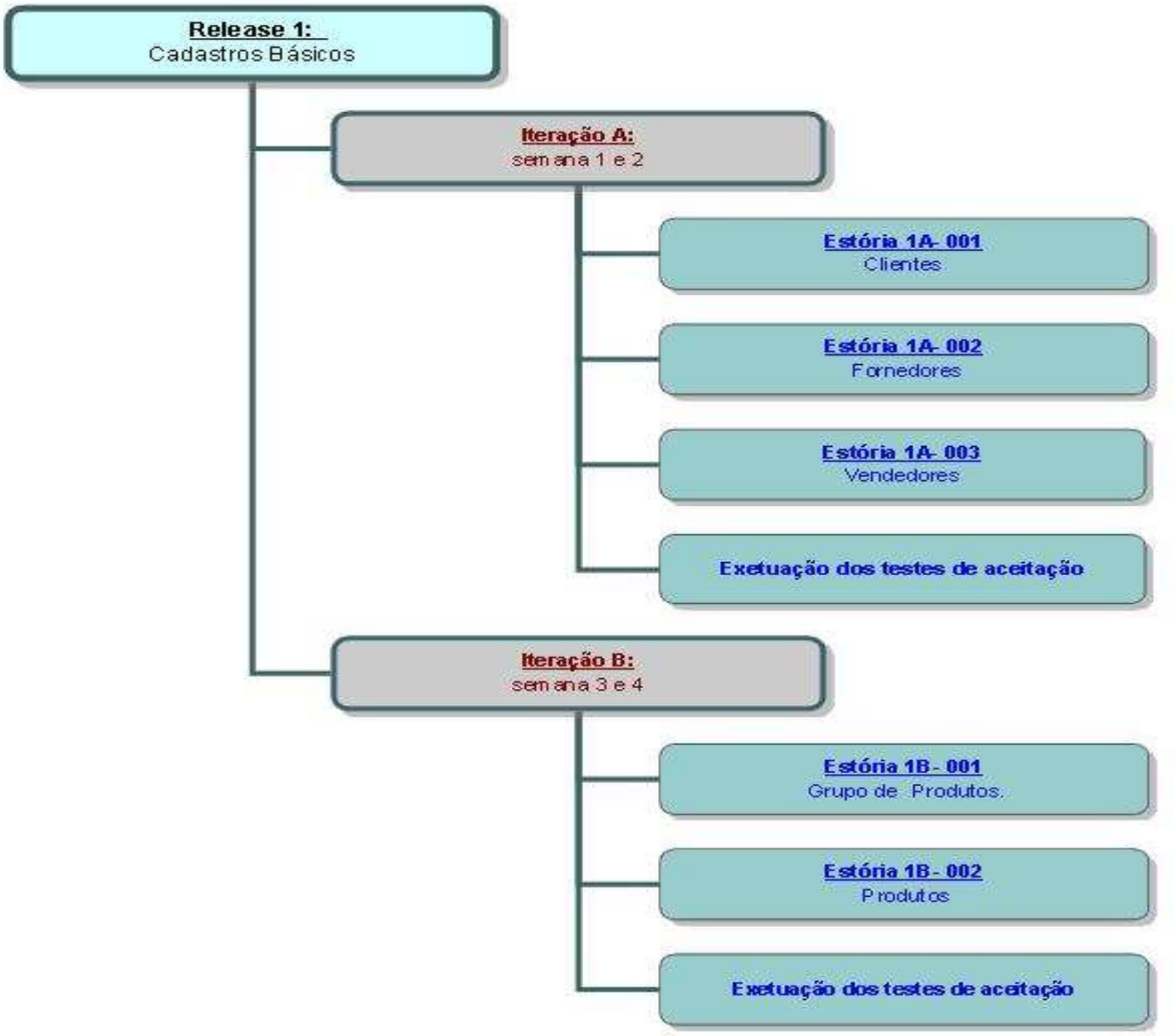

Figura 03 – Estórias, e iterações dentro um release.

Conclusões

Bem caro amigo, espero sinceramente que tenha gostado dessa breve introdução sobre um tema tão excitante e útil para o dia-a-dia no desenvolvimento de software, por isso gostaria de agradecer sua atenção, e convidar-lhe a ler os próximos artigos que darão continuidade a esse tema. Portanto até a próxima.

Referências:

http://www.devmedia.com.br/articles/viewcomp.asp?comp=1498

Artigo: Extreme Programming – Conceitos e Práticas

http://www.devmedia.com.br/articles/viewcomp.asp?comp=1694

Artigo: Implementando Pair Programming em sua equipe

http://www.devmedia.com.br/articles/viewcomp.asp?comp=1432

Artigo: JUnit - Implementando testes unitários em Java – Parte I

http://www.devmedia.com.br/articles/viewcomp.asp?comp=1549

Artigo: JUnit - Implementando testes unitários em Java – Parte Final

http://www.devmedia.com.br/articles/viewcomp.asp?comp=2692

Artigo: O papa-léguas e o coiote - Crônica sobre dois perfis existentes em um projeto de desenvolvimento

http://www.devmedia.com.br/articles/viewcomp.asp?comp=3246

Artigo: O que é melhor? Kung-Fú ou Karatê? - Crônica com alguns devaneios sobre projetos de software

http://www.devmedia.com.br/articles/viewcomp.asp?comp=4073

Artigo: O Pink e o Cérebro - Crônica sobre a difícil arte da simplicidade em projetos de software

http://www.devmedia.com.br/articles/viewcomp.asp?comp=2495

Artigo: JavaDoc – Implementando documentação através

Manoel Pimentel Medeiros
(manoelpimentel.blogspot.com) É Engenheiro de Software, atua em projetos ágeis pela **Heptagon**(SP), entre outras atividades, é diretor editorial da **Revista Visão Ágil**, líder do projeto **BoxSQL** , se dedica a coordenação do grupo de usuários de extreme programming da região Norte (XPNorte) e do **NUG-BR** (NetBeans Users Group - Brasil), além de ser colunista do portal e revista **Java Magazine**, possui também as certificações: Scrum Master, Project Management, Java e Delphi da Brainbench.
.

Contrato de Escopo Negociável

Vinícios Manhães Teles

Projetos XP, como quaisquer outros na área de software, possuem um escopo que define o que deve ser feito. Tal escopo existe antes de o projeto ser iniciado e continua a existir ao longo do projeto até que ele seja encerrado. Entretanto, ao contrário do que é usual, este escopo não é fixado em contrato. Ou seja, caso o cliente perceba a necessidade de fazer ajustes no escopo para que o software leve em conta seu aprendizado ao longo do projeto, ou mudanças nas circunstâncias, ele pode. Em projetos XP, o escopo é revisado freqüentemente para garantir que equipe dedique seus esforços ao que é mais prioritário em cada etapa do projeto.
Como o escopo não é fixo, como o cliente pode saber o que será entregue ao final do projeto, quanto será gasto e qual será o tempo total consumido? Para compreendermos essa questão, vamos começar tratando de um assunto fundamental: **previsibilidade**.

A ilusão da previsibilidade

O que torna o contrato tradicional, de escopo fixo, tão atraente para o cliente? Ele acredita que contará com:

- **Custo previsível**
- **Prazo previsível**
- **Escopo previsível**

Ou seja, acredita que sabe exatamente o que receberá, quando e por que preço. Coloque-se no lugar do cliente. Saber de antemão todas estas informações não parece extremamente atrativo?
Além do mais, olhando pelo lado da empresa que prestará o serviço de desenvolvimento, as perspectivas também são interessantes:

- **Receita previsível**
- **Prazo previsível**
- **Demanda previsível**

Em teoria, a empresa que fará o serviço de desenvolvimento também tem uma situação confortável, pois sabe o que precisa ser feito, quanto irá ganhar e quando estará livre para alocar a equipe em outro projeto. Se é bom para o cliente e é bom para a empresa que desenvolverá o sistema, então onde está o problema? Está na premissa da previsibilidade, que se divide em duas:

1. Cliente sabe exatamente o que deseja no início do projeto
2. Equipe é capaz de estimar com perfeição e entregar o sistema no dia combinado

Vejamos cada uma delas separadamente:

1. Nos projetos de software, o cliente tipicamente não sabe com exatidão o que deseja que o software faça. O que ele costuma saber é o problema que tem em seu dia-a-dia, o qual espera que seja solucionado com a ajuda do software. Durante o projeto, é comum o problema de negócio permanecer basicamente o mesmo, com pouca ou nenhuma alteração. Por outro lado, para qualquer desafio que uma empresa vivencie, existem inúmeras maneiras de solucioná-lo.
Por exemplo, uma grande empresa, com mais de 10 mil funcionários, gostaria de avaliar seus funcionários anualmente para que pudesse identificar deficiências de formação e solucioná-las através de treinamentos apropriados. Avaliar essa quantidade de pessoas leva à necessidade de construir um sistema e a empresa investe na contratação de uma software house para desenvolvê-lo. Existem inúmeras formas e tecnologias que poderiam ser empregadas para construir um sistema desse gênero. Cliente e fornecedor podem acordar uma abordagem, traçando um escopo inicial e, ao longo do projeto, descobrir a possibilidade de simplificar ou facilitar partes do sistema fazendo algo que não havia sido previsto originalmente. Note que, neste caso, altera-se a forma de resolver o problema, mas este permanece inalterado ao longo de todo o projeto. Esperar que o problema se mantenha estável é algo razoável, pois é comum acontecer, mas esperar que a solução necessariamente seja a mesma imaginada originalmente é pouco produtivo, porque raramente acontece. A possibilidade de aprender e aprimorar a solução não é algo ruim. Pelo contrário, é extremamente positivo, podendo significar economia de dinheiro e tempo tanto para o cliente, quanto para a empresa que faz o desenvolvimento.

“Como o cliente aprende ao longo do projeto, ele naturalmente reavalia suas necessidades e prioridades...”

Como se pode notar pelo exemplo, no início do projeto de software, o cliente e a equipe visualizam soluções que representam o escopo inicial do projeto. Ao longo do tempo, à medida que aprendem mais e avançam, é comum surgirem formas alternativas de resolver o problema, às vezes mais simples, mais rápidas de implementar ou com resultados mais expressivos. Se tais alternativas já fossem conhecidas desde o início do projeto, poderiam fazer parte do escopo original, mas como é comum que elas só sejam descobertas ao longo do projeto, é importante que possam ser incorporadas.
Como o cliente aprende ao longo do projeto, ele naturalmente reavalia suas necessidades e prioridades e, portanto, altera o escopo para incorporar seu aprendizado. Quando isso acontece, e quase sempre acontece, a previsibilidade sobre o escopo perde o sentido. Note que, como explicado antes, o problema costuma permanecer basicamente o mesmo ao longo de todo o projeto, portanto é relativamente previsível.

A forma de resolvê-lo, ou seja, o escopo da solução pode e normalmente muda ao longo do tempo. Por isso o escopo não é previsível e fixá-lo freqüentemente se revela uma má idéia.

2. Supondo que o cliente realmente soubesse tudo o que quisesse e não mudasse uma vírgula ao longo do desenvolvimento, bastaria que a equipe fizesse uma boa estimativa para que a previsibilidade sobre o escopo e as demais variáveis do projeto fosse viável. Entretanto, para que a estimativa fosse minimamente acertada, seria preciso que o cliente comunicasse todos os detalhes do sistema para a equipe e que esta compreendesse tudo perfeitamente. Isso é difícil devido a pelo menos dois problemas sérios:
* O cliente normalmente não conta todos os detalhes, até porque não os conhece. Em qualquer sistema que tenha mais que uma meia dúzia de funcionalidades, a quantidade de detalhes tende a ser extremamente elevada. Sistemas são complexos, porque existem muitas combinações que podem gerar os mais diversos tipos de comportamentos, muitos dos quais inesperados. Estes detalhes, inúmeros dos quais não serão ditos pelo cliente, fazem grande diferença no custo e no prazo do desenvolvimento. Portanto, não sabê-los antecipadamente torna o esforço de estimar o sistema bastante limitado.
* Ainda que o cliente apresentasse todos os detalhes seria preciso que a equipe compreendesse tudo corretamente. Como as especificações dos projetos costumam ser expressas de forma escrita, a equipe pode interpretar o que lê das mais diversas formas, o que dá margem para que ela erre feio na estimativa simplesmente por ter interpretado incorretamente os requisitos.
Por tudo o que foi dito, e ao contrário do que muitos gostariam de acreditar, previsibilidade normalmente não passa de uma ilusão na área de software. O cliente finge que acredita e o fornecedor também finge que acredita naquilo que está propondo. Todos estamos habituados a ver que os projetos simplesmente não saem como o previsto e estatísticas, como as produzidas pelo Standish Group, vêm confirmando isso há décadas. Então, o que fazer?

Que tipo de previsibilidade podemos esperar?

Em desenvolvimento de software, é importante que clientes e desenvolvedores compreendam que:

a. Previsibilidade de escopo é inviável na maior parte dos casos
b. Escopo fixo, ao invés de representar previsibilidade, prejudica os envolvidos, especialmente o cliente

Sobre a. eu já comentei, portanto, passemos para o item b.
Quando o cliente opta por um escopo fixo, está apostando que não aprenderá nada ao longo do projeto e que nada diferente ocorrerá em seus processos de negócio. Raramente isso é verdade. O cliente aprende e as empresas convivem cada vez mais com ambientes de negócio que avançam com rapidez e demandam mudanças em seus projetos de software. Portanto, optar por um escopo fixo significa correr um risco, bem elevado, de que o software final não atenda às reais necessidades do cliente. Embora isso já seja suficientemente grave, não é tudo.
Segundo as estatísticas, mais de 60% das funcionalidades de um software comercial típico jamais são utilizadas quando colocadas em produção. Ou seja, se a equipe de desenvolvimento produzir exatamente o que está no escopo original, ela provavelmente estará produzindo uma grande quantidade de funcionalidades que jamais serão usadas. Em outras palavras, mais de 60% do investimento poderá parar na lata-de-lixo porque não irá gerar nenhum valor para o cliente, por não ser usado.
De fato, a melhor forma de administrar um projeto de software é rever permanentemente as prioridades do cliente e assegurar que apenas funcionalidades essenciais, isto é, que serão usadas de verdade, sejam colocadas no sistema. Só é possível saber quais são estas funcionalidades ao longo do desenvolvimento, enquanto o cliente aprende com o software que está sendo construído, especialmente quando o desenvolvimento é iterativo, como no caso do Extreme Programming. Ser iterativo significa receber software funcionando a cada final de iteração, o que permite utilizar as funcionalidades, aprender com elas e rever que funcionalidades ainda poderão trazer valor para o projeto com base no feedback concreto daquelas já implementadas. Perder essa oportunidade, fixando um escopo no início, é extremamente prejudicial para o próprio cliente.

O que é um contrato de escopo negociável?

É um contrato que se baseia na premissa (bastante realista) de que não existe previsibilidade sobre o que será feito no software. Embora seja possível haver previsibilidade em relação aos gastos e ao tempo. Como se poderá observar, é também uma forma de alinhar os interesses do cliente e da empresa que irá desenvolver o software.
Existem quatro variáveis essenciais que precisam ser abordadas em qualquer contrato de desenvolvimento:

- **Custo**
- **Prazo**
- **Escopo**
- **Qualidade**

O tradicional contrato de escopo fixo determina claramente qual será o custo, o prazo e o escopo. A qualidade pode até ser abordada, mas normalmente é sacrificada assim que o prazo aperta. Já o contrato de escopo negociável segue outro caminho. Visto que as quatro variáveis são conflitantes até certo ponto, não é possível fixar todas elas. Uma alternativa é fixar:

- **Custo**
- **Prazo**
- **Qualidade**

Assim, permite-se que o escopo absorva as incertezas do projeto. Neste caso, o cliente continua sabendo o quanto irá gastar, bem como quanto tempo o projeto irá durar. O que ele não sabe com exatidão é o que irá receber. Mas, na verdade, ele não sabia disso no caso do contrato de escopo fixo, ele apenas tinha a ilusão de saber, a ilusão da previsibilidade do escopo. Portanto, ele não perdeu absolutamente nada,

Por sua vez, a qualidade pode ser tratada no contrato determinando que o projeto seja desenvolvido utilizando práticas que assegurem elevados padrões de qualidade, tais como:

- **Desenvolvimento Orientado a Testes**
- **Programação em Par**
- **Refatoração**
- **Código Coletivo**
- **Desenvolvimento Iterativo**
- **Integração Contínua**

As práticas do XP são organizadas de modo a assegurar que as prioridades sejam respeitadas e revistas periodicamente, bem como altos padrões de qualidade sejam mantidos. Desenvolvendo software de forma iterativa, ou seja, entregando mais funcionalidades a cada iteração semanal, o cliente tem inúmeras oportunidades de rever as prioridades, bem como avaliar o trabalho da equipe. Isso permite que ele aprenda ao longo do projeto, incorpore o seu aprendizado ao sistema e decida o que tem valor ou não e, portanto, deve ser implementado ou não.
Esta decisão é o que permite que ele atinja a data alvo com um software que tenha, no mínimo, as funcionalidades que mais irão gerar valor para ele. Isto é, se não for possível desenvolver todas as funcionalidades, queremos assegurar que as funcionalidades que ficarem de fora do sistema sejam aquelas que produziriam menos valor, porque as mais valiosas já são implementadas no início do projeto. Esta filosofia costuma ser mais valiosa para o cliente, porque ao final ele tem um software que atende as suas necessidades e prioridades reais e não às que ele achava que tinha no início do projeto.

O que fazer se o cliente contratar uma equipe inadequada para o projeto?

Em qualquer contrato, independente do tipo, existe o risco de que a equipe não corresponda às expectativas do cliente. É importante que o cliente conheça a capacidade real da equipe o quanto antes e, com base na sua observação, possa decidir se deseja ou não continuar com ela. Em contratos de escopo fixo, especialmente em projetos tradicionais com desenvolvimento sequencial, o cliente só saberá se fez uma boa escolha após o projeto já ter avançado muito, pois o código demora a ser produzido, portanto, o feedback demora a aparecer.
No XP, o cliente começa a receber funcionalidades prontas e pode utilizá-las já ao final da primeira semana. E terá ainda mais funcionalidades, na semana seguinte e assim sucessivamente. Isto fornece inúmeras oportunidades para avaliar e decidir se deseja ou não continuar com a equipe, o que ajuda a administrar o risco do projeto. Então, na prática, como seria o texto de um contrato de escopo negociável?
Primeiramente, ainda antes de o projeto começar, é preciso estimar o tempo necessário e a quantidade de pessoas a serem alocadas na equipe. Para isso, pode-se fazer um levantamento inicial de funcionalidades como em qualquer projeto. Não existe mágica neste sentido. A empresa que fará o desenvolvimento e o cliente terão que conversar sobre a visão do que o futuro sistema deverá fazer e quais serão suas funcionalidades. Com base nisso, pode-se estimar o número de pessoas recomendável, bem como o prazo desejado.
O custo do projeto naturalmente é proporcional à quantidade de tempo e pessoas alocadas ao projeto. Será que todas as funcionalidades imaginadas no escopo original estarão prontas no prazo combinado? A equipe de desenvolvimento não sabe, assim como o cliente também não sabe. Alías, ele nem sabe se serão estas as funcionalidades ou se elas serão modificadas ao longo do tempo. Ao invés de buscar previsibilidade e uma estimativa perfeita, o que se espera neste momento é identificar valores que sejam razoáveis, tanto para o tempo, quanto para o custo e o número de pessoas. Feito isso, o contrato pode seguir o exemplo abaixo.
"O projeto terá a duração de oito meses com iterações semanais. A equipe terá seis desenvolvedores ao custo de R$ 60 mil/mês. Cliente e equipe devem discutir as funcionalidades a serem desenvolvidas a cada início de iteração. Caberá à equipe de desenvolvimento indicar o número de funcionalidades possível de serem entregues por iteração. Os pagamentos serão mensais e o contrato é revisado a cada dois meses, quando o cliente tem a opção de permanecer com a equipe de desenvolvimento ou encerrar o projeto sem ônus."

E se os desenvolvedores fizerem corpo mole?

O contrato é simples. Indica quantas pessoas serão alocadas, por quanto tempo e qual o custo delas por mês. Parece ser basicamente um contrato de locação de mão-de-obra com pagamento baseado em utilização de horas. Mas não é, pois há um detalhe fundamental para se compreender a filosofia deste modelo de contratação: o cliente tem opções de saída. Isto é, de tempos em tempos, o cliente pode cancelar o contrato sem nenhum ônus, ou seja, sem ter que pagar multas contratuais.
Neste exemplo, ao final dos primeiros dois meses, o cliente já terá recebido software relativo a oito iterações semanais. Ou seja, terá tido oito oportunidade de utilizar o trabalho concreto produzido pelos desenvolvedores. Isso representa informação suficiente para saber se a equipe está caminhando com um ritmo adequado ou não. Ao longo de todo o projeto, a cada dois meses, o cliente pode decidir se mantém ou troca a equipe de desenvolvimento.
Errar na contratação de uma equipe é mais comum do que se imagina. Entretanto, infelizmente estes erros são descobertos tardiamente, quando o custo de repará-los é muito elevado. Errar é natural e não devemos temer isso, pois aprendemos e evoluímos com os erros. O que devemos temer é levar tempo demais para descobrir que erramos. Isso é o que o XP e o contrato de escopo negociável procuram evitar. Iterações semanais permitem descobrir cedo se erramos na contratação. Contratos de escopo negociável permitem reverter uma contratação inadequada cedo, ainda no início do projeto, quando poucos recursos foram investidos. Assim ajuda a administrar o risco do cliente, além de alinhar objetivos.

Alinhando interesses

No exemplo de contrato apresentado, ao começar um projeto, a equipe de desenvolvimento só terá garantia do faturamento dos primeiros dois meses, pois ao final desse tempo o cliente pode mandá-la para casa se não estiver satisfeito. Entretanto, ela naturalmente deseja participar do projeto nos outros seis

meses subseqüentes, de modo que possa elevar seu faturamento. Sendo assim, ela tem razões para querer fazer um execelente trabalho e com agilidade, para que o cliente queira renovar o contrato a cada dois meses. Por outro lado, ela não tem nenhuma razão para rejeitar as mudanças propostas pelo cliente, porque o escopo não está fixado e o pagamento não está atrelado a ele. Sendo assim, o cliente pode alterar funcionalidades ao longo do projeto sem que isso afete a capacidade da equipe de cumprir o contrato. Os interesses ficam alinhados. Todos querem um trabalho que atenda da melhor forma possível às necessidades do cliente porque isso beneficiará tanto a equipe de desenvolvimento quanto o cliente.

No modelo de contrato tradicional, onde o escopo é fixado, alterações sugeridas pelo cliente tendem a ser rejeitadas pela equipe de desenvolvimento ou cobradas com valores elevados. Afinal, como o escopo é fixado no contrato, mudanças efetuadas nele afetam a capacidade da equipe de cumprir o contrato. Sendo assim, contratos deste tipo naturalmente levam cliente e desenvolvedores a se tornarem adversários em um jogo no qual o cliente tenta maximizar a quantidade de funcionalidades que recebe e os desenvolvedores tentam minimizar o esforço e as funcionalidades. Além disso, tais contratos são mais caros, pois as equipes de desenvolvimento prevêem que os clientes farão alterações no escopo e, por isso, cobram mais que o necessário, de modo a cobrir esse risco.

No contrato de escopo negociável, no pior dos casos, se o cliente não gostar do trabalho, poderá interrompê-lo sem maiores problemas. Neste caso, ele pode ter perdido tempo e dinheiro, mas a perda é limitada a dois meses de trabalho (no exemplo apresentado). Em contratos tradicionais, com desenvolvimento seqüencial, onde as funcionalidades demoram para aparecer, o cliente normalmente só descobre que a equipe está fazendo um trabalho ruim próximo ao fim do contrato, quando o custo (direto ou indireto) de interromper já é muito maior.

A questão da confiança

No contrato de escopo negociável a empresa que faz o desenvolvimento fica relativamente vulnerável pelo fato de o cliente poder suspender o contrato após os primeiros dois meses se não estiver gostando do trabalho ou não tiver confiança na equipe. O grande problema neste caso é a questão da confiança. Quanto mais confiança o cliente tiver, menor serão as chances de ele interromper. O que se pode fazer para alcançar uma confiança elevada? Criar um histórico de credibilidade e aproximar o cliente ao máximo dos desenvolvedores.

O XP recomenda que o cliente participe do desenvolvimento e isso é essencial. Se estiver envolvido no dia-a-dia do projeto, conhecerá melhor o trabalho da equipe, seu ritmo, suas deficiências, seus pontos fortes, seu empenho e comprometimento. Ou seja, se a equipe estiver realmente dedicada a fazer um bom trabalho ele conseguirá notar isso. Não existe nada mais poderoso para estabelecer uma relação de confiança entre cliente e desenvolvedores que aproximar ambas as partes tanto quanto possível e criar um histórico de credibilidade. Isso ocorre naturalmente quando a equipe de desenvolvimento entrega consistentemente as funcionalidades acordadas em cada iteração, o que é possível utilizando-se as práticas do XP que, entre outras coisas, ajudam a criar um ritmo de trabalho ágil e impõem inúmeras proteções ao software, de modo a evitar perdas de tempo com correções desnecessárias ou excessivamente demoradas.

O custo reduzido dos contratos de escopo negociável

Contratos de escopo negociável são, por definição, mais baratos que contratos de escopo fixo. Por que? Porque a software house que oferece um contrato de escopo fixo precisa incorporar o risco de que a equipe tenha interpretado o escopo de forma incorreta, ou que o cliente altere o escopo, o que pode levar a necessidade de mais pessoas ou mais tempo de desenvolvimento. Em ambos os casos, o fornecedor tem uma perda financeiro. Para cobrir este risco, o fornecedor cobra um valor acima do seu custo real para cobrir os custos que surgirão caso os riscos se materializem. No caso do contrato de escopo negociável, não é necessário cobrar por este risco, porque o escopo é negociado e discutido diversas vezes ao longo do projeto. O escopo não está vinculado ao contrato, portanto, não há risco de o fornecedor deixar de cumprir com o contrato por um erro de interpretação da equipe ou alterações no escopo efetuadas pelo cliente ao longo do projeto. Por essa razão tais contratos podem e devem ser mais baratos.

Adoção

Aqui na Improve It, já utilizamos contratos de escopo negociável com alguns clientes e eles representam uma grande vantagem competitiva. Se a sua empresa é uma software house, provavelmente também poderá se beneficiar deste modelo de contrato. Basta compreendê-lo e propô-lo aos clientes. Não será necessariamente fácil, mas pode ser um caminho interessante para reduzir custos, riscos e criar um relacionamento mais harmonioso com os clientes.

Contratos de escopo negociável representam uma mudança cultural. Ao vendê-los, é necessário expor todas as questões abordadas anteriormente, em especial a falta de previsibilidade e o aprendizado do cliente ao longo do projeto. Estas informações servem para justificar a necessidade de uma mudança no modelo de contrato. Se o cliente perceber esta necessidade, fica mais fácil introduzir o contrato de escopo negociável como uma alternativa que resolva o problema. Por fim, apresentar o custo do contrato de escopo negociável e compará-lo com o custo de um contrato de escopo fixo ajuda a convencer o cliente de que esta é uma boa alternativa. Se precisar de ajuda, fale conosco. Temos satisfação em trabalhar com esse modelo de contrato e ajudar outras empresas a adotá-lo.

Vinícius Manhães Teles

- Fundador da Improve It
- Mestre e bacharel em Informática pela UFRJ
- Autor do primeiro e único livro escrito no Brasil sobre Extreme Programming
- Atua como coach em projetos XP e mentor em treinamentos na mesma área
- Pioneiro na adoção do Extreme Programming no Brasil
- Professor de Extreme Programming no curso de graduação de Ciência da Computação da UFRJ

Mostra-me tua equipe, e eu te direi quem és!

Alexandre Magno Figueiredo

Empresas e projetos

O que significa o fracasso de um projeto para uma empresa? De uma forma simplificada poderíamos dizer que significa a perda de dinheiro, reputação e coragem. Dinheiro por motivos óbvios, reputação pelo desgaste que a equipe sofrerá com a queda, e coragem pelo fato de que a alta diretoria da empresa se sentirá apreensiva para investir em próximos projetos. Talvez, por este motivo pudemos perceber que nos últimos anos as companhias têm ficado mais atentas para a prática qualificada do gerenciamento de projetos. Na maioria das verticais de negócio, vemos um crescente investimento na capacitação e qualificação dos gerentes de projetos, mais ainda em ferramentas, metodologias, certificações e outros. No entanto, no fim de cada projeto, continuamos a assistir o mesmo filme de sempre: “Caça às Bruxas'. Ou foi culpa do gerente, ou da diretoria, ou da falta de um maior investimento, ou dos programadores, ou da inexistência de treinamentos, ou – como é na maioria dos casos – a culpa foi do cliente(interno ou externo), que não sabia exatamente o que queria! (Esse assunto geraria facilmente um novo artigo, mas convido-os a visitar a lista agile-brasil, onde o tema já foi bastante debatido). Enfim, a temporada de caça está aberta, e a pergunta continua sendo a mesma: **de quem é a culpa?**

Ritual de fim de projeto

A resposta para essa pergunta pode ser obtida facilmente seguindo o seguinte ritual:

1.Convoque todos (exatamente todos) os envolvidos no projeto para uma reunião.

2.Quando todos estiverem presentes, solicite que se levantem e, após o uuumm...doooiss..trêêêss, repitam: **A CULPA É NOSSA!** Pronto, a pergunta está respondida.

Onde está a equipe?

A resposta obtida no “ritual” que citei, dificilmente será aceita pela maioria dos participantes da reunião, afinal, se eles não foram uma equipe durante o projeto inteiro, não é agora – hora de encontrar culpados – que se tornarão uma. Mas será que se tivéssemos uma verdadeira equipe o resultado final seria diferente? Com certeza! Gerentes de projetos tem começado a perceber repetidamente algo em comum em projetos bem sucedidos: equipes de verdade, e – consequentemente - de alto desempenho. Mas o que é uma equipe de alto desempenho?

Equipes de alto desempenho

Um equipe de alto desempenho é aquela que possui um objetivo bem definido e compartilhado por todos os membros do projeto. É uma equipe onde todos estão entusiasmados com suas atividades, querem vencer juntos. Os membros dessas equipes são criativos e ousados na busca da qualidade, são dedicados, acordam na segunda-feira com vontade de ir para o projeto, pois se divertem no trabalho, e se divertem juntos, querem sempre fazer mais e melhor. Todos querem participar de uma equipe dessa, pois sabem que além de uma experiência profissional inigualável, isso será um divisor de águas na sua carreira.

Agora, quando isso não existe, o que você tem são clientes, diretores, gerentes, analistas, programadores e outros, cada um de um lado, com um “crachá” diferente.

Qual sua missão nesse projeto?

Tive uma experiência no passado, quando estava gerenciando um pequeno projeto que – para a diretoria – não estava gerando os resultados esperados. Quando menciono para a diretoria, é porque para nós – gerente, analistas e programadores – estava tudo aparentemente “nos trilhos”. Mas me frustou ouvir de meu diretor que não, que os resultados que eles precisavam não eram apenas cronograma afinado e resultados aparentes. Fui encorajado por ele a fazer um “acompanhamento” junto à nossa equipe para realizar a seguinte pergunta a todos: “Qual sua missão nesse projeto?”. Enquanto eu realizava esse trabalho pude perceber onde estava o problema e, ao finalizar, cheguei à mesa do diretor e disse: “Você estava certo, temos que mudar algo em nosso time”. O algo a mudar não eram pessoas, ferramentas, métricas ou metodologias, mas sim a forma com a qual a “equipe” enxergava o projeto. No acompanhamento, programadores responderam que sua missão no projeto era “programar”, analistas “analisar e modelar” e por ai vai. Ninguém mencionou palavras relacionadas ao sucesso do projeto, a resultados, ao objetivo maior, ao algo que se busca – aí sim – programando, modelando, gerenciando, etc. Portanto, o projeto estava fadado ao fracasso.

“Um equipe de alto desempenho é aquela que possui um objetivo bem definido e compartilhado por todos os membros...”

Como formar equipes de alto desempenho?

Uma série de fatores influenciam no sucesso ou fracasso dessas equipes, infelizmente não há uma receita, e é bom que tenhamos em mente que nem sempre isso será possível. Segue algumas atitudes que podem ajudar na formação dessas equipes:

- Siga os valores do manifesto ágil

- Tenha claro os objetivos do projeto. É extremamente importante que a expectativa de todos os envolvidos seja a mesma.

- Valorize a comunicação, montando inclusive um ambiente favorável às conversas espontâneas.

- Providencie recursos, principalmente econômicos, para que a atenção – e preocupação – da equipe esteja focada no projeto.

- Valorize a criação.

- Valorize as opiniões dos membros da equipe.

- Aceite falhas.

- Encoraje a ousadia.

- Torne o trabalho prazeroso e divertido.

- Prefira qualidade à quantidade, ou seja, desenvolver menos com qualidade é melhor que mais sem qualidade.

- Crie um nome, escudo, slogan...enfim, algo que identifique e diferencie a equipe.

- Encoraje os conflitos construtivos.

- Preocupe-se com o objetivo coletivo e não individual.

Alexandre Magno Figueiredo
É líder de projetos de software onde utiliza principalmente metodologias e processos ágeis. Atua na área de software há mais de 14 anos, já tendo participado de projetos de variadas dimensões de lead time, escopo e investimento. É Certified Scrum Practitioner, possuindo ainda certificações dos fornecedores IBM e Borland, e dos grupos OMG e PMI. Magno é o fundador do grupo Scrum-Brasil.

Scrum e as armadilhas das reuniões diárias

Alexandre Magno Figueiredo

Uma das principais práticas do Scrum é a reunião diária (Daily Meeting). Essas reuniões se resumem em uma atividade de quinze minutos(pode ser um pouco mais ou menos, mas só um pouco) que deve ser realizada todos os dias. Durante estas reuniões os membros do time do projeto devem responder às seguintes perguntas:

- O que fiz desde a última reunião diária?

- O que farei até a próxima?

- Estou tendo algum impedimento?

Quando realizo apresentações de Scrum, é comum perceber o quão simples esta atividade parece ser para a maioria das pessoas, porém, por traz de uma simples "reuniãozinha", a reunião diária esconde o termômetro do seu projeto...e não é preciso dizer o que pode acontecer quando um termômetro é mal utilizado, certo?

“Muitas equipes tem confundido os quinze minutos da reunião diária com momentos de desabafo...”

Reunião diária não é "hora do café"

Algumas equipes, no decorrer do projeto, passam a achar que a reunião diária é algo bastante trivial, sem portanto exigir a necessidade de uma formalização com horário e local fixo. A consequência disso é que, com o passar do tempo, reuniões diárias passam a ser realizadas na sala de cafézinho, no "fumódromo" ou até na padaria. Na verdade a equipe passa a substituir o propósito do “Daily Meeting” para "conversas informais de quinze minutos sobre o projeto". As reuniões diárias devem sim ser realizadas no mesmo local e horário (pelo menos enquanto durar o sprint corrente). Isso é importante!

Uma sugestão de locais ideais para a realização das reuniões seriam: salas de reunião ou de treinamento, ou em algum local com espaço suficiente para acomodar toda a equipe. A sala do chefe é uma péssima opção. Quanto ao horário, li em muitos artigos e/ou livros que o horário da manhã é o ideal para as reuniões diárias do Scrum, o que faz até sentindo mas, ná prática, eu realmente não vi exatamente com estes olhos. Trabalhei com reuniões no início do dia e ao final dele, e em ambos tive prós e contras. Pela manhã todos os participantes estão descansados e prontos para um novo dia de trabalho, isso é bom! Mas pela manhã sempre(e sempre mesmo) tem alguém atrasado(trânsito, chuva, filhos, etc.), ou seja, em pouquíssimas vezes você consegue realizar a reunião com toda a equipe, e isso é muito ruim! No fim do dia estão todos um pouco(ou muito) cansados, o que diminui a empolgação durante as reuniões, isso é ruim! Em compensação, é bem mais comum você ter toda a equipe presente, e isso é bom! Então, na minha opinião, analise o cenário cultural envolvido(empresa, projeto e equipe) e escolha uma das duas opções.

Reunião diária não é "conversa sobre futebol"

Se você pretende abrir as reuniões diárias para participação de pessoas apenas envolvidas com o projeto(chickens), deixe bem claro ao fazer o convite que o mesmo dá apenas direito a participação como "ouvinte". Apenas as pessoas que estejam realmente comprometidas com o projeto(pigs) devem falar durante a reunião diária. Deixar "chickens" falarem durante a reunião é permitir que pessoas que não estão tendo um acompanhamento dia-a-dia do projeto opinem sobre o seu andamento. Isto provavelmente causará um mal-estar entre "pigs" e “chickens” e, pior ainda, fará com que os quinze minutos acabem antes que você perceba.

Reunião diária não é "conversa sobre a relação"

Muitas equipes tem confundido os quinze minutos da reunião diária com momentos de desabafo e resolução de problemas. Muitas vezes já tive que interromper algum programador quando este falava sobre os impedimentos que estava encontrado: “Estou com um problema com a framework XYZ, pois lá tem a classe Zummble que preciso instanciar...mas não consigo fazer isso devido a esta classe herdar da JJJ que possui a implementação da interface U. Se eu fizer a instância direta cairei em um problema de redeclaração de método, blá, blá, blá”. Bem, este simples parágrafo geraria debates técnicos que, com certeza, ultrapassariam os quinze minutos da reunião. Qual seria a forma ideal de reportar este impedimento na reunião diária? Algo tipo: “Estou tendo dificuldades com a framework XYZ e creio que eu esteja precisando de um suporte/ajuda com isso para poder ter um ritmo mais adequado". Ponto final! Com isso, após a reunião, é papel do Scrum Master providenciar uma solução para o problema do programador, seja convocando uma reunião técnica, solicitando suporte junto ao fabricante, organizando uma programação em par com alguém mais experiente na framework...enfim.

Obviamente, você não pode sair cortando e proibindo todos os assuntos que emerjam na reunião e não estejam ligados às três perguntas, até porque é natural que esses outros assuntos apareçam. Eu particularmente costumo usar a estratégia do *Rat Hole* da FDD durante as reuniões diárias. O *Rat Hole* nada

mais é que um papel de *flip-chart* ou mesmo um espaço em um quadro branco, que receberá entradas toda vez que algum assunto pertinente, mas que não deveria ser aprofundado durante esta reunião, emergir . Após a finalização do meeting, o Scrum Master define quando e de que forma tais assuntos serão discutidos.

Reunião diária não é um "julgamento"

Este talvez seja o erro mais freqüente nas reuniões diárias. Os times, principalmente aqueles compostos por membros que não estão habituados a fazer parte de equipes auto-gerenciadas, insistem em usar os quinze minutos da reunião para reportar-se **AO Scrum Master**, e na maioria das vezes se justificando por algum atraso ou problema do gênero. A reunião diária foi criada principalmente para o time, e não para o Scrum Master. E é **AO time** que cada membro deve se reportar. A reunião diária é o que nos ajuda a termos sempre uma visão atualizada do seguinte: de onde viemos? onde estamos? para onde estamos indo? E essa talvez seja a sua grande valia! Se você usar esses preciosos quinze minutos diários para penalizar membros da equipe por atraso, ouvir justificativas, dar lição de moral ou coisas do gênero, você não conseguirá utilizar o termômetro.

Infelizmente os times são os que mais demoram para entender isso. Se você, como Scrum Master, não conseguir avaliar se seu time está capitando bem a finalidade das reuniões diárias, experimente durante um dia não estar presente na empresa no horário da reunião – não avisando nada antecipadamente. Se ao retornar eles lhe informarem que não realizaram a reunião, afinal você não estava presente...você tem problemas! Eles realmente ainda não entenderam que a reunião diária é feita para eles.

Portanto entenda - e faça o time entender - que as reuniões diárias:
- Nos ajudam a analisar nossa velocidade/ritmo;
- Fazem com que todo o time esteja a par do que está acontecendo em todo o projeto;

- Identificam impedimentos;

- Integram diariamente o time;

- E, principalmente, nos fornecem um termômetro sempre atualizado de "Como está o nosso projeto?"

Alexandre Magno Figueiredo
É líder de projetos de software onde utiliza principalmente metodologias e processos ágeis. Atua na área de software há mais de 14 anos, já tendo participado de projetos de variadas dimensões de lead time, escopo e investimento. É Certified Scrum Practitioner, possuindo ainda certificações dos fornecedores IBM e Borland, e dos grupos OMG e PMI. Magno é o fundador do grupo Scrum-Brasil.

As Cinco Doenças do Gerenciamento de Projetos

Causa Nº 1: Multi-Tarefa Nociva

Adail Muniz Retamal

Como é possível completar mais projetos, mais rápido, sem sacrificar a qualidade ou o escopo, quando seus recursos já estão mais do que sobrecarregados?

Seus projetos sofrem de alguns desses efeitos indesejáveis?

- Atrasados
- Recursos sobrecarregados
- Mudanças em excesso (devido aos longos prazos do projeto)
- Recursos não disponíveis quando necessários (mesmo quando prometidos)
- Prioridades mutáveis, retrabalho

Este artigo define as cinco razões relacionadas ao comportamento humano, responsáveis por seus projetos estarem atrasados e que, se você não abordá-las, continuarão a atrasá-los.

Nós descobrimos, através de anos de prática e pesquisa, que projetos sofrem de destinos similares. Examinando nossa biblioteca de mais de cem livros sobre gerenciamento de projetos e liderança descobrimos que o livro mais antigo e o mais novo, sobre gerenciamento de projetos, identificam as mesmas reclamações. Por mais de 50 anos os projetos têm sofrido dos mesmos efeitos. Por quê? O que poderia estar causando tal fracasso, universalmente, por tanto tempo? Caso você acredite que seu trabalho é diferente, também descobrimos que não importa qual tipo de projetos você gerencia. Projetos de energia, militares, tecnologia da informação, construção e dúzias de outros campos sofrem de destino idêntico.

Cinco razões, contra as quais seus projetos lutam, foram identificadas:

1. Multi-tarefa nociva
2. Síndrome do Estudante
3. Lei de Parkinson
4. Dependência entre tarefas
5. Matemática do gerenciamento de projetos, onde 2+2=5

Você deve se preocupar com essas cinco causas porque elas atrasam os benefícios e resultados do projeto para sua empresa e seus clientes. Essas causas também atrasam o fluxo de caixa de um projeto finalizado e permitem que sua equipe e seu cliente encontrem uma janela de oportunidade maior para fazer mudanças que ameacem o próprio projeto. Imagine a redução de pedidos de alteração se seu projeto fosse completado duas vezes mais rápido. Lembre-se, se você continuar fazendo o que sempre fez, continuará obtendo o que sempre obteve, projetos atrasados.

Causa Nº 1: Multi-Tarefa Nociva

Você ou sua equipe enfrentam constantemente prioridades que mudam, fazendo com que interrompa uma tarefa e trabalhe em outra? Tem alguém esperando pela saída de sua tarefa para que possa fazer o trabalho dele/dela? Esta é a definição de multi-tarefa nociva. Dito isto, nem toda multi-tarefa é nociva. Quando ninguém está esperando pela sua saída não há nada de errado em comutar entre várias tarefas.

Por que fazemos multi-tarefas? Para alguns de nós é por causa do tédio de trabalhar em uma coisa por vez. Nossa mente exige estimulação mais alta e, portanto, continuamente mudamos de assunto. Freqüentemente a culpada é a má priorização. Nos pedem para iniciar várias tarefas simultaneamente e cada uma delas possui um "cliente" esperando por sua saída. Cada cliente quer que a tarefa *dele* progrida e constantemente pergunta "já terminou?", forçando-nos a comutar repetidamente para a tarefa dele para que algo seja feito e reportar o progresso. Enquanto estamos trabalhando nesta tarefa, outros clientes pedem o status de suas respectivas tarefas. Este ciclo nos força a comutar tarefas repetidamente. Naturalmente, ao trabalhar numa tarefa você não está fazendo progresso em nenhuma outra. Se seus clientes contam com uma entrega rápida, eles levarão seus negócios para outro lugar. Para alguns clientes só o progresso já é suficiente. Não é necessário que seja rápido, desde que esteja sendo feito. Porém, mesmo se você tiver sorte o bastante para ter tais clientes, qual é o impacto em você e no seu negócio?

"Assim, a multi-tarefa sempre faz com que uma tarefa demore mais do que deveria..."

Qualquer quantidade de tempo não trabalhado numa tarefa significa que a tarefa está sendo atrasada mais do que seria se você se dedicasse ao seu término. Assim, a multi-tarefa *sempre* faz com que uma tarefa demore mais do que deveria. Entre outros fatores que se somam está o tempo de raciocínio que leva para "entrar no trilho" para tornar-se criativo. Para tarefas como engenharia, programação e redação esse tempo pode ser uma parte significativa do tempo total da tarefa quando se faz multi-tarefa. Para o trabalho manual isto pode incluir o tempo de ajuste da máquina, preparação das ferramentas e equipamentos apropriados e colocá-los de volta em seus lugares. Existem algumas tarefas onde o tempo de ajuste é desprezível e não é um fator, mas essas são poucas no mundo do trabalho intelectual. Estimativas indicam que o ajuste, ou tempo de raciocínio, pode igualar ou mesmo exceder o tempo real da tarefa, para tarefas altamente cognitivas. Um exemplo inclui escrever este artigo. Quando eu me distancio

dele para fazer outra coisa qualquer e depois volto, preciso ler o segmento inteiro de novo para descobrir onde eu estava e o que estava pensando quando parei. Isto toma um tempo extra, que poderia ser devotado a mais escrita.

Pense como a multi-tarefa afetaria você numa fila de caixas da mercearia. Imagine que, em vez de atender uma pessoa por vez, o processo de pagamento no caixa incluísse a varredura de um produto de cada pessoa na fila, e então fosse repetido. Se há apenas uma pessoa na fila, o tempo necessário para o pagamento seria apenas o tempo necessário para varrer seus produtos, pegar seu dinheiro e empacotar suas mercadorias. Entretanto, se após você chegar na fila e um ou mais de seus itens tivessem sido registrados, outra pessoa entra na fila e, em vez de completar seu pedido, o caixa pegar um item desta outra pessoa, registrá-lo, e depois pegar um de seus itens, registrá-lo, e repetir. Agora demorará duas vezes mais para você completar sua compra. Enquanto o caixa está registrando seus itens e, simultaneamente, o da pessoa atrás de você, outra pessoa entra na fila. Agora o caixa pega um item seu, depois um da próxima pessoa, e depois um da nova pessoa, e repete. O que acontece quando outra pessoa entra na fila? Como você pode ver, quanto mais pessoas entrarem na fila, mais demorará para você completar sua transação. Você compraria nesta loja mais de uma vez? Não. Então, por que você faz isso com sua equipe e com seus clientes? A forma mais rápida de completar uma transação é começá-la e fazê-la até terminá-la. Você pode, então, concentrar-se na tarefa e no cliente. É mais rápido e fornece o melhor serviço ao cliente. Ninguém reclama, a menos que a fila fique muito grande. Quando comutamos de tarefas, o risco de problemas com a qualidade também aumenta. Nos esquecemos do que foi feito e do que não foi feito. Nós corremos para retornarmos a outra tarefa. Passamos por cima de pequenos detalhes em nosso ajuste. A pressão de clientes irados adicionam estresse ao trabalho, nos tornando menos satisfeitos e sujeitos a negligenciar um bom serviço. O envolvimento da gerência aumenta, para lidar com clientes "importantes" ou altamente impacientes, e a aceleração (despacho) começa a tornar-se uma forma de lidar com eles. Quanto mais e mais clientes começam a exigir um serviço mais rápido, a gerência começa a focar em métodos complicados de priorização para satisfazer a todos e manter os empregados focados em "fazer a coisa certa".

A multi-tarefa nociva força as pessoas a dar estimativas maiores do que o necessário para as tarefas. Se você sabe que *não* lhe será permitido iniciar uma tarefa, trabalhar nela até terminar e depois prosseguir para a próxima tarefa, você será forçado a dar uma estimativa muito maior sobre quanto tempo levará para completar a tarefa. Se você sabe que levará dois dias para completar uma tarefa, mas também sabe que será interrompido, incluirá o tempo de interrupção na estimativa. Agora, uma tarefa de dois dias é estimada em 10 dias. Seu cliente esperará dez dias? Você decide. Seu cliente ficaria mais motivado a fazer negócio com você se você prometesse dois dias em vez de dez? Imagine a vantagem competitiva da promessa de menor duração. Imagine o melhor ambiente de trabalho criado para os membros da sua equipe quando você elimina os métodos complicados de priorização, a constante aceleração (despacho), os clientes zangados e a constante vigilância da gerência.

Nem toda multi-tarefa é nociva. Como saber a diferença? Lembre-se, multi-tarefa nociva é quando uma tarefa está sendo atrasada e a pessoa a quem você deve o resultado está esperando você terminar. Na verdade, a multi-tarefa nociva pode atrasar o término de todo o projeto. Porém, se ninguém estiver esperando pelo resultado então não é multi-tarefa nociva. Pode não ser eficiente, mas pode não significar muito. Por exemplo, lhe pedem para encher 100 envelopes e colocá-los no correio para amanhã. Já que o correio passa apenas uma vez ao dia e já é muito tarde para colocá-los hoje, não importa se você dobrar todos os papéis primeiro e depois envelopá-los (multi-tarefa), ou dobrar um papel e envelopá-lo e depois prosseguir para o próximo (sem multi-tarefa). Entretanto, se o carteiro estiver chegando a qualquer minuto e você tem um item crítico para enviar, seria insensato fazer multi-tarefa simplesmente para ser eficiente. Nesse caso a tarefa requer dobrar, envelopar, selar e enviar o lote, sem interrupção.

Algumas funções são guiadas por multi-tarefa. Nem toda função deveria eliminar a multi-tarefa. Por exemplo, uma secretária, um chefe de cozinha e muitas outras funções exigem a movimentação de muitas partes. Isto é multi-tarefa nociva? Não necessariamente. Devido à natureza muito curta da tarefa, e de tempos de espera embutidos na própria tarefa, o atraso é usualmente tolerável. Você pode precisar responder várias chamadas telefônicas e colocar pessoas na espera. Este atraso não é geralmente um problema até que se torne excessivo. Eventualmente o cliente determina um limite na quantidade de tempo que você pode permitir nos atrasos por multi-tarefa, antes que ele desligue e faça negócio em outro lugar. Todos nós gostaríamos que nossas chamadas fossem respondidas imediatamente, nosso problema fosse abordado sem ficar esperando, e então concluiríamos nossa chamada. Porém, devido à curta duração da transação, nós estamos dispostos a aceitar uma certa quantidade de demora.

A simulação de tarefas demonstra os efeitos da multi-tarefa nociva. Em simulações de projetos reais (usando a simulação com três projetos, de Tony Rizzo) realizadas em meus seminários, os estudantes realizam três projetos com multi-tarefa. Os efeitos criados durante este exercício são caos, confusão, montes de ordens e muitas atividades adicionais de "gerenciamento". É bastante estressante. Quando eles terminam eu os desafio a fazer duas vezes mais projetos, seis no total, em menos tempo do que os três originais. Eles nunca acreditam que isto é possível. Eles também não querem enfrentar mais esse tanto de estresse. Entretanto, uma vez que removemos a multi-tarefa nociva, eles sempre fazem duas vezes mais projetos em menos tempo e sem o caos, sem gritaria, e muito menos estresse. A única diferença entre os dois eventos é a multi-tarefa nociva. Quando sua equipe está fazendo multi-tarefa isso exige uma sobrecarga considerável de gerenciamento. Alguém precisa manter o registro do que está sendo trabalhado, o status, o tempo previsto de término, e atualizar o cliente repetidamente. Cada pedaço dessa sobrecarga pode ser eliminada. Se sua experiência é parecida com isso, você tem muito a ganhar removendo esse obstáculo. Se você deseja conseguir fazer duas vezes mais, no mesmo período de tempo, e reduzindo o estresse, pare com a multi-tarefa. A figura abaixo fornece um exemplo gráfico dos resultados da multi-tarefa versus a não multi-tarefa.

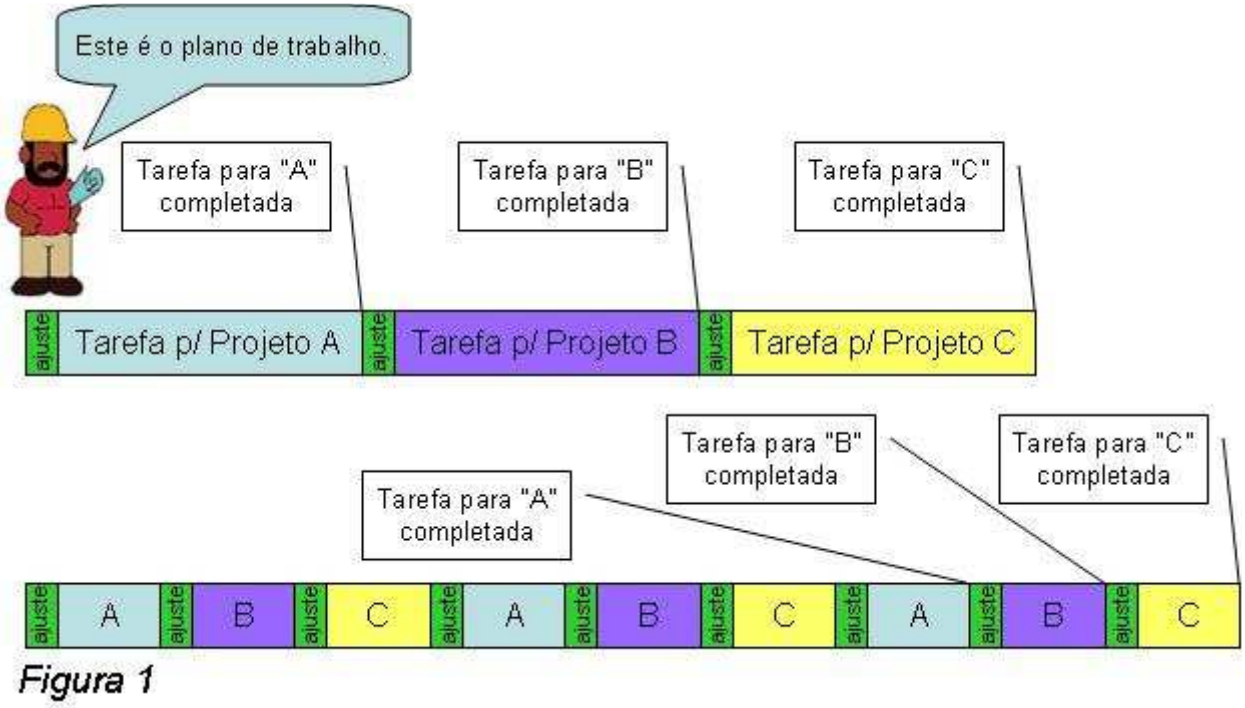

Figura 1

Na **Figura 1**, note o término antecipado devido à remoção do tempo de ajuste (raciocínio). Também note o prazo de entrega de cada tarefa. Mesmo se os dois cenários demorarem a mesma quantidade de tempo total (zero tempo de ajuste/raciocínio), a vantagem de não fazer multi-tarefa é significativa. Se alguém estiver esperando pelos resultados da Tarefa-A antes que possa realizar sua tarefa, é fácil ver que sem a multi-tarefa a próxima tarefa pode começar consideravelmente mais cedo. E mais ainda, note que quando se faz multi-tarefa as três tarefas são completadas em rápida sucessão. Se os resultados de todas as três tarefas forem para o mesmo recurso, o destinatário agora herdou a carga da multi-tarefa. Isto cria uma pilha de trabalho completado que se move correnteza abaixo, assim como um excesso de trabalho em andamento. E tem mais, pode-se ter perdido tempo enquanto o próximo recurso esteve esperando pela saída. Ele pode ter ficado "ocupado" com trabalho, para ter certeza de que não seria pego sem ter o que fazer (e arriscar ser demitido). O tempo gasto nessa "ocupação com trabalho" não avançou o projeto e pode até mesmo ter contribuído para atrasá-lo.

Geralmente me perguntam sobre o "tempo morto" durante as tarefas. "Supõe-se que eu fique sentado, fazendo nada, quando eu chegar num ponto de uma tarefa onde eu estiver esperando por outros?" Não. Lembre-se, é apenas multi-tarefa nociva se alguém está esperando pelos seus resultados. Isto é, alguém está esperando pela sua saída para realizar o trabalho dele/dela. Por exemplo, se você está cozinhando e coloca o assado no forno, você está esperando o forno completar a tarefa dele. Você está livre para se mover para outra tarefa enquanto espera. Entretanto, esteja pronto para continuar imediatamente a tarefa anterior quando o forno tiver acabado, caso contrário, você queimará o assado.

Adail Muniz Retamal ***(*Tradutor)***
É diretor da Heptagon Tecnologia da Informação Ltda, empresa de consultoria e treinamento focada na aplicação da Teoria das Restrições em geral, e da Corrente Crítica em particular, à Engenharia de Software, metodologias ágeis de gerenciamento e desenvolvimento de software, atuando como catalisador da mudança organizacional, além de ajudar as organizações a construírem sua estratégia e tática para alinhar seus esforços com suas metas. Adail é Engenheiro Eletricista/Eletrônico, atuou como consultor, instrutor e arquiteto de soluções para a Borland Latin America por 4,5 anos. Lecionou em universidades públicas e privadas. É palestrante, articulista e está escrevendo um livro. Adail pode ser contatado em adail@heptagon.com.br.

Allan Elder ***(*Autor)***
É presidente da No Limits Leadership, Inc., uma empresa de consultoria dedicada a ajudar as organizações a entregar mais projetos, mais rápido, atráves da liderança eficaz. Allan trabalhou como diretor de GSI (Gerenciamento de Sistemas de Informação) para a segunda maior empresa de seguros corporativos e a maior empresa de segurança na Califórnia. Allan foi certificado como PMP, é um "Jonah" na Teoria das Restrições, possui bacharelado em Telecomunicações, mestrado em Gerenciamento de Projetos e Ph.D. em Organização e Gerenciamento. Além de seu trabalho em consultoria, Allan é o principal instrutor de gerenciamento de projetos para a Universidade da Califórnia, Irvine, prestou consultoria e lecionou para a UCI Graduate School of Business, e foi um Examinador Sênior para o California Award for Performance Excellence (CAPE) por três anos. Allan pode ser contatado em aelder@nolimitsleadership.com.

Revista Visão Ágil

Apoio:

HEPTAGON
TECNOLOGIA DA INFORMAÇÃO
www.heptagon.com.br

tc4digital
www.tc4digital.com

aXmagno
www.axmagno.com

Fratech
SOLUÇÕES EM SISTEMAS
Treinamentos | Mentoring | OutSourcing
www.fratech.net

IMPROVE IT
www.improveit.com.br