



Estratégias para Projetos de Software



Tabuleiro de Projetos

Uma breve crônica sobre estratégias em projeto de software, através de peças de xadrez

Um cardápio de

Metodologias Ágeis

Uma visão geral dos principais processos ágeis.

**Desenvolver ou não Desenvolver,
eis a questão.**

Uma analogia, sobre a mecânica de projetos.

As 5 Doenças do Gerenciamento de Projetos

Causa n° 2:
Lei de Parkinson

Desenvolvimento Orientado a Testes

Veja os principais pontos sobre TDD

E mais:

Humor de Projetos, Visão Ágil pelo Mundo,
Referências Ágeis e Veio da Enquete



Sprint Backlog

Conheça os temas selecionados para essa edição(iteração).



News
Página 05



Veio da Enquete

Página 06



Visão Ágil pelo Mundo

Página 07



As 5 Doenças do Gerenciamento de Projetos

Causa n° 2:
Lei de Parkinson
Página 08



Desenvolvimento Orientado a Testes

Veja os principais pontos sobre TDD
Página 14



Humor de Projetos

Página 18



Um cardápio de Metodologias Ágeis

Uma visão geral dos principais processos ágeis.
Página 19



Referências Ágeis

Página 23



Desenvolver ou não Desenvolver, eis a questão.

Uma analogia, sobre a mecânica de projetos.
Página 24



Tabuleiro de Projetos

Uma breve crônica sobre estratégias em projeto de software, através de peças de xadrez
Página 26



Editorial

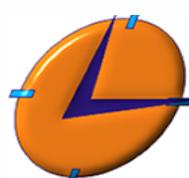
É com muito prazer que iniciamos essa segunda edição da **Revista Visão Ágil**, agora, estamos cada vez mais nos consolidando como o canal de informações sobre **processos ágeis** no Brasil.

Após a primeira edição, tivemos vários feedbacks positivos e várias pessoas interessadas em colaborar com artigos em nossa revista, por isso agora, estamos trazendo alguns nomes fortes do cenário ágil através de artigos inéditos sobre assuntos quentes, como por exemplo a segunda parte das **5 Doenças do desenvolvimento**, **Desenvolvimento Orientado a Testes**, **Cardápio de Metodologias Ágeis**, **Desenvolver ou não desenvolver, eis a questão**, e o **Tabuleiro de Projetos**.

Além de trazermos algumas seções inéditas como **Visão Ágil no Mundo**, **Humor de Projetos**, **Referências Ágeis** e **Veio da Enquete**.

Portanto, caro amigo agilista, espero que goste dessa segunda edição, pois foi feita com muito carinho, focando em qualidade, objetividade e clareza nas informações. Boa Leitura.

Manoel Pimentel Medeiros
Diretor Editorial



Revista **Visão Ágil**

Equipe Visão Ágil

Diretor Editorial: Manoel Pimentel Medeiros

Diretor de Marketing: Alexandre Magno Figueiredo

Acessoria Administrativa: Manuella Bulcão Medeiros

Atendimento ao leitor

e-mail: manoelp@gmail.com

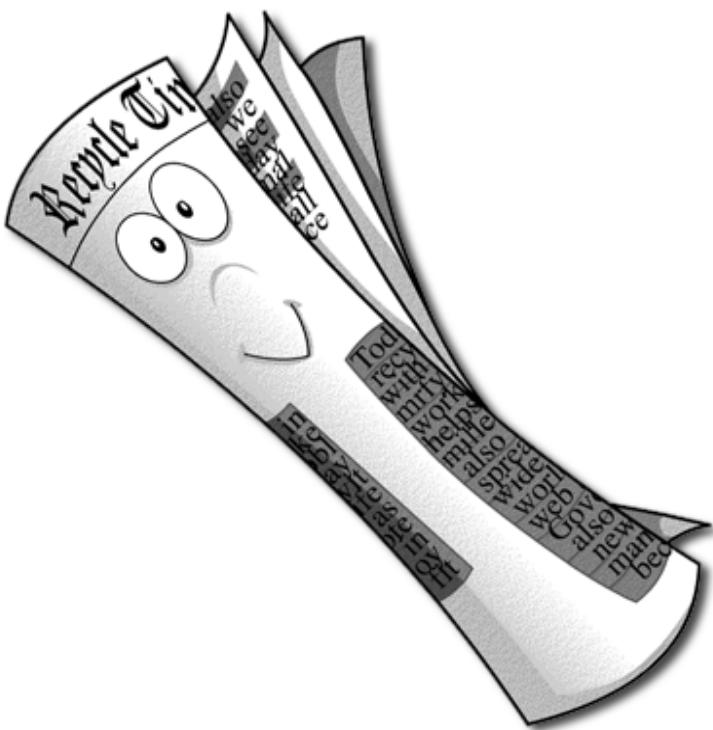
site: www.visaoagil.com

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.



News

Veja aqui as novidades
do mundo ágil

Evento JavaBrasil 2007,
Confira maiores informações em:
www.javabrasil.org

Encontro sobre Scrum em Londres
Novembro em Londres
informações: www.axmagn.com

Evento BorCon 2007
<http://info.borland.com.br/borcon>

Workshop
Struts 2 in Agile em Novembro
www.fratech.net

Workshop
Java for web in Agile em Novembro
www.fratech.net

Coding dojo floripa dia 01/11
<http://dojofloripa.wordpress.com>

Veio da Enquete

Acompanhe o resultado
da enquete do mês



Neste mês, ocorreu a enquete com seguinte pergunta: “**Quais processos ágeis você já usou em seus projetos?** (Escolha todas que se aplicam)”. Confira quais eram as opções e como ficou o resultado dessa enquete.

Respostas

Opções	Votos	%	1 resposta
ASD	0	0	<input type="text"/>
DSDM	0	0	<input type="text"/>
FDD	4	6	<div style="width: 6%;"><div style="width: 100%;"></div></div> <input type="text"/>
Scrum	22	33	<div style="width: 33%;"><div style="width: 100%;"></div></div> <input type="text"/>
XP	29	44	<div style="width: 44%;"><div style="width: 100%;"></div></div> <input type="text"/>
Outras	10	15	<div style="width: 15%;"><div style="width: 100%;"></div></div> <input type="text"/>

Visão Ágil pelo Mundo

Nossa revista se fortalecendo
e ampliando seu público

Desde o lançamento da primeira edição da revista **Visão Ágil**, tivemos várias vitórias que se mostraram em números. Primeiro foi o nosso grupo de usuários, que está quase atingindo o total de 500 membros. Em seguida, segundo dados fornecidos pelo **Google Analytics**, tivemos uma grande procura em nosso site, inclusive com acessos vindos de outros países como: United States, Israel, Portugal, Germany, Canada, Mozambique, United Kingdom, Belgium, Japan, Switzerland, Turkey, Argentina, New Zealand. Confira esses países no mapa abaixo.



2,233 visits came from 15 countries/territories



As 5 Doenças do Gerenciamento de Projetos

Causa n° 2:
Lei de Parkinson

Tradução de **Adail Muniz Retamal**

A **multi-tarefa nociva** faz os membros da equipe embutirem maior segurança nas estimativas de tarefas. Vamos examinar mais detalhadamente o tópico da estimação das tarefas e avaliar a validade e o prejuízo causado ao inflacionar a quantidade de segurança nas estimativas de duração. Quando atribui-se uma tarefa para uma pessoa, a primeira pergunta é: “**Quanto tempo você vai demorar?**” Você concorda que as pessoas têm uma tendência a incluir “proteção” na estimativa da duração, para acomodar fatores externos como “**Murphy**” e multi-tarefa? Pense numa tarefa recente que você estimou. Que nível de certeza você ofereceu? Foi 100%? Provavelmente não, pois algo poderia acontecer e impedir você até de completar a tarefa, tal como uma morte e, portanto, 100% de certeza não é alcançável. Que tal 50%? Mesmo neste nível de certeza você estaria atrasado em 5 de cada 10 vezes. Talvez sua oferta tenha sido próxima a 90%. Isto significa que 9 em cada 10 vezes você está no prazo. Isto parece razoável? Que nível de certeza você exige de sua equipe? Você exige que suas estimativas sejam precisas toda vez? Ou você está bem com atrasos em 5 de cada 10 vezes? A maioria dos gerentes quer que as pessoas forneçam estimativas “precisas”, o que não faz sentido porque uma estimativa, por definição, é apenas uma aproximação.

As 5 Doenças do gerenciamento de Projetos

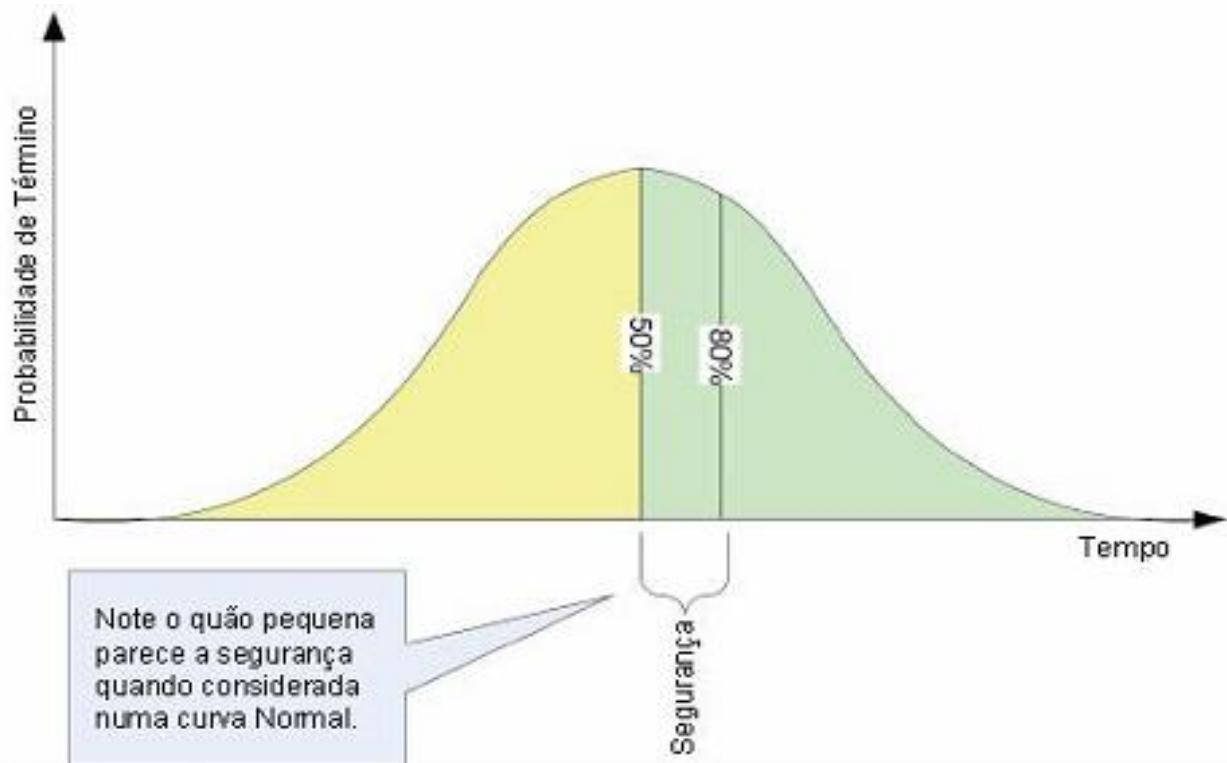


Figura 2

Intelectualmente nós entendemos que estimativas não são exatas. Mesmo assim, nós as exigimos. Por quê? Há uma crença subjacente de que se podemos determinar precisamente o tempo para cada tarefa, e nos assegurarmos que cada tarefa seja terminada “no prazo”, então o projeto terminará no prazo. Porém, nós também sabemos que o objetivo não é terminar cada tarefa no prazo, mas completar o projeto no prazo. Segurança ‘normal’ A realidade do trabalho do projeto é que a incerteza existe e, portanto, o tempo das tarefas não pode ser determinado, apenas estimado

O resultado da exigência de estimativas precisas é que as estimativas de duração são convertidas em compromissos. Assim, para fornecer uma estimativa realista devemos levar em conta todas as coisas que podem impactar na duração da tarefa, embutindo segurança.

Quando uma “pequena” segurança é adicionada às estimativas elas não são consideradas como irrazoáveis porque, mentalmente, nós adicionamos a segurança considerando uma distribuição normal do tempo.

Numa distribuição normal, 50% do tempo está de um lado da média e 50% do tempo está do outro lado da média. Assim, avançar de 50% de probabilidade para 80% não parece ser significativo (veja Figura 2).

Porém, tempos de tarefas não são “normais”. De fato, não existe tal coisa como “normal”. A distribuição normal ocorre apenas na matemática, não na vida real. Devido ao Teorema do Limite Central, se incluirmos amostras suficientes, eventualmente teremos a distribuição “normal”. Um exemplo bobo é se eu colocar um pé num balde de água fervente e outro pé num balde de água gelada, na média, eu devo me sentir confortável.

As 5 Doenças do gerenciamento de Projetos

Tempos de tarefas não seguem uma curva normal. Segurança 'real' Em vez disso, eles começam em algum lugar além do zero (toda tarefa deve tomar alguma quantidade de tempo) e então a probabilidade de completá-la como prometido aumenta rapidamente, e logo começa a diminuir numa longa cauda (**veja a Figura 3**). Quando comparar os dois gráficos verá que a "pequena" segurança embutida é, na realidade, bastante grande. Quando maior a incerteza da tarefa, mais a cauda cresce. O resultado é uma tarefa com aproximadamente metade de sua duração estimada sendo segurança.

O único a adicionar segurança é o membro individual da equipe? Nunca. Posteriormente, o gerente toma as tarefas estimadas com segurança e adiciona a sua própria segurança. Em cada nível de gerência mais segurança é adicionada (ver Figura 4). Essa segurança adicional desnecessariamente prolonga a data de término do projeto e, de fato, não protege contra a incerteza. Algumas vezes outra doença ocorre, isto é, todo mundo inflaciona suas estimativas porque sabem que o nível acima irá cortá-las. Já que o recurso não sabe o quanto será arbitrariamente cortado, ele "chuta" o quanto inflacionar a duração. Continuando, já que o próximo nível não tem idéia de quanta segurança foi adicionada, ele corta "chutando" o quanto de segurança ele acha que foi adicionada. O processo todo é ridículo.

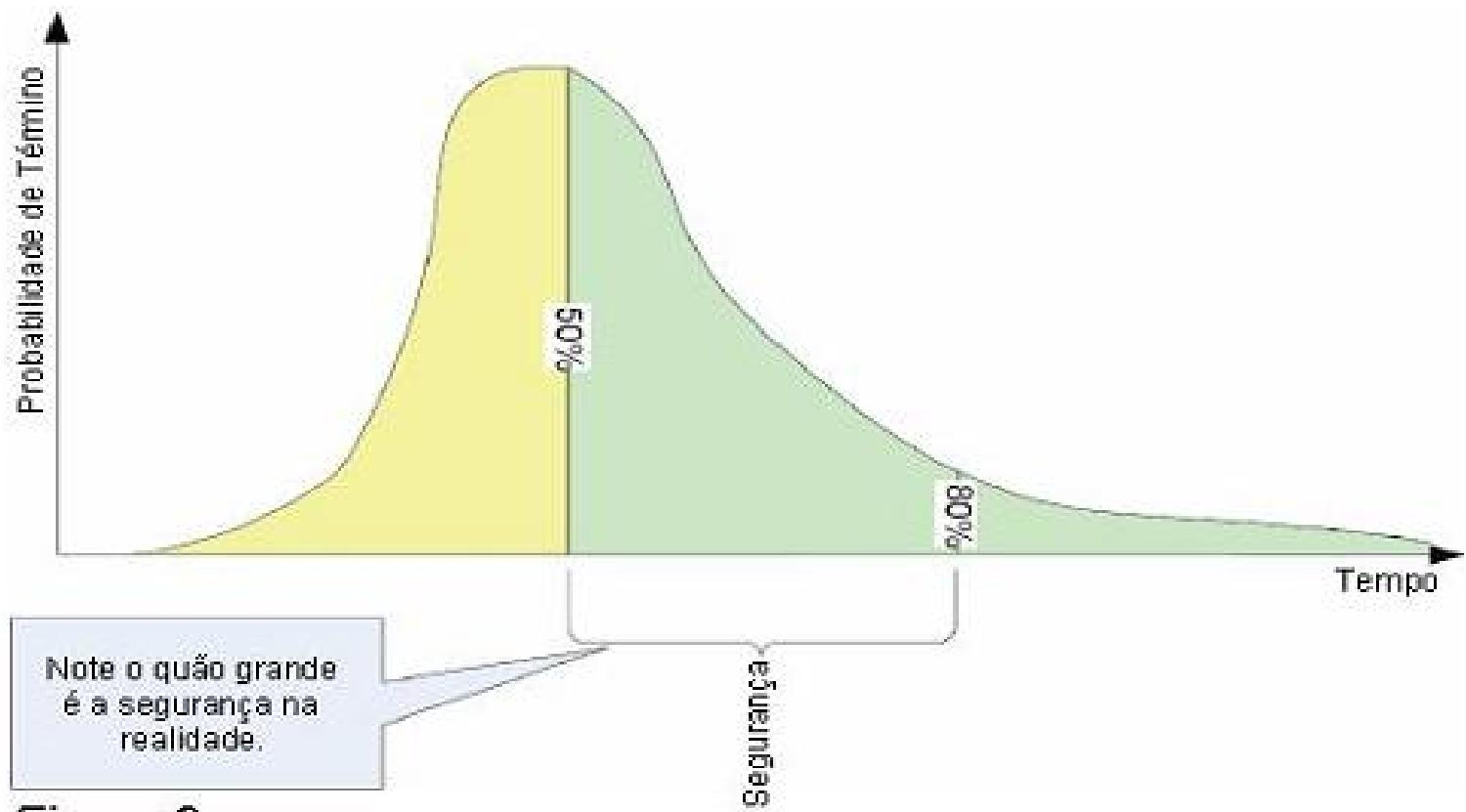


Figura 3

As 5 Doenças do gerenciamento de Projetos

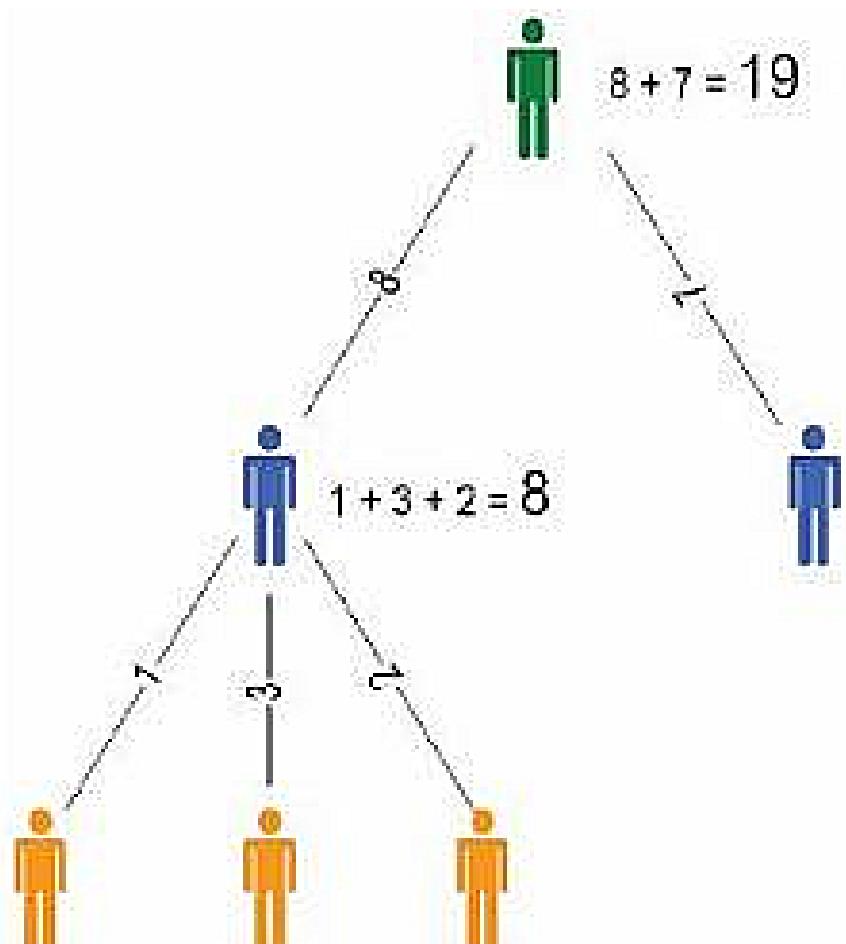


Figura 4

Se há tanta segurança embutida em cada tarefa, por que as tarefas continuam a atrasar? Não deveriam elas terminar no prazo ou antes? Se você concorda que a segurança é embutida para levar em conta o desconhecido (Murphy), e que Murphy não ataca toda tarefa como previsto, então a maioria das tarefas não deveria terminar no prazo. Deveria terminar antes. Apenas uma tarefa ocasional, que teve tudo imaginado durante a estimativa, e aí teve uma sorte muito pior, deveria atrasar. Se as tarefas não estão terminando antes na maior parte do tempo, então a segurança está sendo perdida.

Mesmo uma tarefa que termina no prazo é inaceitável, pois aproximadamente metade do tempo estimado estava reservado para eventos que nunca ocorreram (Murphy).

O mais notável ainda é que as pessoas se esforçam para cumprir o pedido absurdo de fornecer estimativas precisas. Por quê? Você concordaria que a maioria das pessoas quer ser considerada confiável? Se é assim, isto significa que nós tentamos cumprir nossos compromissos, certo? E se isso é verdade, o resultado é que nós adicionamos segurança e lutamos para impedir que ela seja removida por outros.

Se temos segurança extra embutida, que não foi necessária, então usamos esse tempo extra para fazer um trabalho melhor, em vez de reportar um término antecipado.

Afinal, se exigimos 6 semanas e entregamos em 4 semanas, qual será a “recompensa” por entregar antes, na próxima vez que pedirmos 6 semanas para uma tarefa? Nossa segurança será cortada. Isto pode fazer com que falhemos em nosso compromisso, nos atrasando e nos tornando não confiáveis. Este fenômeno é tão predominante que possui seu próprio nome:

Lei de Parkinson. Esta lei expressa que: “O trabalho se expande para preencher o tempo disponível.” Agora você deve concordar que as pessoas realmente adicionam segurança, mas ela não está sendo usada apropriadamente. Sua prova é que a maioria das tarefas não terminam antes, como seria esperado.



**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, através 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.

Java Brasil 2007

Conferência Ágil para profissionais Java

Muito além do cafezinho



*maratonas
negócios
agile
opensource
tecnologia*



novidades
cultura
workshops
palestrantes internacionais
mobilidade

Venha para o Java Brasil 2007

e fique por dentro das maiores novidades em tecnologia Java!

Dias 2, 3 e 4

Novembro de 2007

Espaço de Convenções Hotel Nacional Inn Campinas/SP

INFORME-SE: (19) 3119.6880

www.javabrasil.org

- 36 horas de evento
 - 70 palestras em 5 salas
 - Coffee Break
 - Profissionais consagrados
 - Workshops e palestras com:
 - Ian Roughley • Autor do livro "Starting Struts 2" e desenvolvedor ativo do Struts 2
 - David Hussman • Agile Coach há mais de 7 anos, co-autor do livro "Agile in the Large"

Organizações



Patrocinio:





Desenvolvimento Orientado a Testes

Por **Victor Hugo Germano**

Os tempos são de mudança. Não se pode mais pensar em um modelo de negócios que não sofra influência das áreas econômica, política ou estratégica. Nossa modelo de desenvolvimento deve estar pronto para responder rapidamente às alterações que por ventura afetem um projeto.

Nesse ambiente complexo, em que requisitos nem sempre são definidos e a única certeza que temos é a da mudança, devemos assumir uma postura pró-ativa que dê ao desenvolvimento a chance de responder rapidamente à alteração do processo de negócio.

Assim surgem os modelos evolucionários de desenvolvimento, sendo o TDD (Test Driven Development), uma maneira de garantir o progresso sustentável do software frente às incertezas que cercam seus requisitos e objetivos futuros. Trazendo um modelo de simplicidade e clareza regendo todo o ciclo de vida do produto, o Desenvolvimento Orientado a Testes pode ser uma ótima opção de garantia da qualidade e robustez no desenvolvimento de software.

Desenvolvimento Orientado a Testes

Antes de tudo, deixem-me esclarecer uma coisa: este não é um artigo sobre testes. Este artigo é sobre um método de desenvolvimento que nos guia para um código mais robusto e passível de manutenção. Softwares complexos não precisam possuir um código incompreensível, e quanto maior a complexidade, maior ainda será o impacto positivo gerado pelo TDD.

Desenvolvimento Orientado a Testes é uma técnica de desenvolvimento de software que envolve a definição de um caso de teste, ou mesmo a implementação de um código de teste, antes de iniciar a implementação de uma funcionalidade ou método. A implementação deve ser apenas para fazer o teste passar. Desta maneira, pode-se ter um feedback rápido sobre o progresso da implementação. Formalizado primeiramente através dos trabalhos de Kent Beck em sua introdução ao eXtreme Programming, foi atribuída ao TDD grande responsabilidade, sendo definido como o cerne de todo o Desenvolvimento Ágil. Fora do ambiente Ágil, entretanto, TDD torna-se um sub-processo que acaba fazendo parte de todo o processo de Desenvolvimento de Software.

As três Regras do TDD

No blog *Uncle bob*, existe uma explicação simples por meio de três regras:

- Nenhum código de produção será escrito a não ser que seja para fazer um teste unitário passar.
- Apenas o essencial será codificado para que um teste unitário falhe; e erros de compilação são erros
- O único código de produção escrito será o suficiente para fazer o teste unitário passar.

Ken Beck(2003) atribui a estas regras uma mudança drástica no comportamento dos indivíduos da equipe:

- Desenvolvimento orgânico, com código funcionando provendo feedback rápido entre as decisões
- Escrever os próprios testes e não ter que esperar 20 vezes ao dia para que outra pessoa os escreva para você
- O ambiente de desenvolvimento deve prover respostas mais rápidas para pequenas mudanças (i.e. a necessidade de compiladores e suites de teste mais robustas)
- A arquitetura do sistema torna-se altamente coesa e componentizada, sendo ainda menos acoplada (devido à característica altamente normalizada da técnica), tornando os testes mais fáceis de serem executados.



Desta maneira, viramos o desenvolvimento tradicional do avesso. Criando software através de passos menores, podemos manter o controle do código escrito, garantindo que qualquer funcionalidade adicionada deva ser previamente projetada e testada. Na verdade, um programador utilizando TDD à risca, recusa-se a escrever uma nova funcionalidade sem que ao menos um teste tenha sido criado para validar tal incremento de valor.

E os benefícios vão muito além. A cada hora inúmeros testes são criados, e nenhuma nova funcionalidade é adicionada sem que todos os testes anteriores estejam sendo executados perfeitamente sem falhas. Isso cria a garantia de que caso um problema ocorra, ele poderá perfeitamente ser reproduzido criando mais alguns testes, e ainda assim, se uma funcionalidade nova for inserida, pode-se garantir que ela não afetará outros pontos do sistema, caso todos os testes estejam validados.

Neste caso, os testes servem não apenas para garantir que o código esteja perfeitamente funcionando, mas obriga que desenvolvedores idealizem o funcionamento prévio das funcionalidades, isto é, cria a necessidade da análise do problema durante o desenvolvimento, fazendo com que a compreensão e o modelo emerja da contínua tentativa de fazer um teste passar. Aonde chegaremos com isso? O design da aplicação é criado, especificado e materializado através de uma ferramenta que possibilita a automatização de verificações:

o teste.

Este modelo força os desenvolvedores a pensarem de forma desacoplada, favorecendo a componentização. Tendo a intenção de testar um módulo isoladamente, é necessário criá-lo de maneira componentizada, forçando-os a pensar em otimização de código, reutilização e em simplicidade

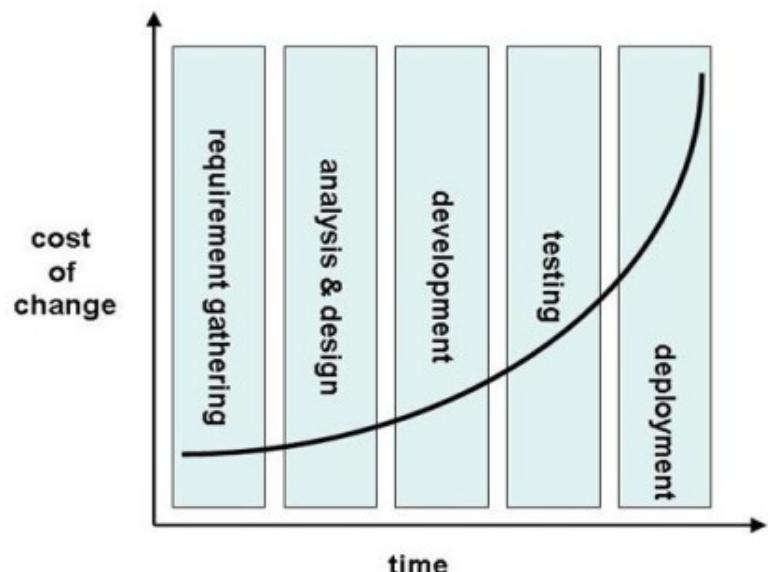
Ampliando a Qualidade de Software

Tradicionalmente temos a fase de testes de software ocorrendo no momento posterior à implementação, seguindo o modelo já ultrapassado, mas ainda utilizado, o *cascata*. Com esta postura, existe a idéia de que a equipe de testes deve varrer o sistema à procura de erros e bugs. E se o real entendimento da área de testes fosse outro? Que tal interpretar uma área de testes como um local onde apenas será verificado se os erros realmente não existem, assumindo que o desenvolvimento já está empenhado em evitar que tais erros aconteçam, deixando para a área de testes a função de garantir o alinhamento entre Requisitos e Software em funcionamento.

O principal ganho do modelo TDD não é descobrir erros, mas evitar que eles sejam criados.

A expressão mais utilizada atualmente é "build quality in", isto é, garante-se o feedback imediato sobre a implementação, e em função disso amplia-se a confiança no código e a garantia de que o sistema realmente alcance os requisitos propostos.

Num modelo cascata, o custo da alteração de requisitos cresce exponencialmente à medida que o sistema evolui entre as fases de Análise, Implementação, Teste e Implantação. Temos assim, o quadro abaixo, retirado do artigo "Custo da Mudança" de Barry Boehm (em inglês):



Surge uma observação importante: sendo o custo tão alto, movendo-se a fase de testes para a esquerda, talvez houvesse uma chance de reduzir o custo de uma mudança, já que antes de passar entre as fases estariam validando o que foi feito na fase anterior.

Neste ponto, o desenvolvimento evolucionário TDD acaba se favorecendo, pois permite que sejamos mais flexíveis diante de mudanças nos requisitos. Ao invés de levar muito tempo em todas as fases do desenvolvimento, entregando - ou não - ao final do processo um produto, nos encontramos em uma situação onde se pode demonstrar partes completamente funcionais do sistema ou funcionalidades muito antes. E, ao fazer isso, o custo de mudança torna-se imensamente menor.

Independentemente da opção de se utilizar XP ou qualquer outro método Ágil, o TDD torna-se uma opção para qualquer modelo de desenvolvimento. Automatizar testes, tentar especificar funcionalidades de maneira incremental, garantir a consistência do código desenvolvido e ainda a possibilidade de calcular o impacto gerado por bugs de maneira fácil são os pontos fortes da adoção.

Documentação

Encaremos um fato: desenvolvedores dificilmente são adeptos de manter e ler documentações, preferindo, normalmente, os códigos. Como programador, me agrada, antes de tudo, quando inicio o trabalho com uma nova biblioteca, verificar qual o código utilizado para acessar essa nova biblioteca. E não há nada de errado nisso. Testes unitários bem escritos possuem esse propósito, provendo uma especificação funcional do código, e em decorrência disso são uma ótima referência técnica.

Da mesma forma, a definição de o que os stakeholders esperam do sistema pode ser automatizada através de testes. Este processo de especificação de requisitos em forma de testes é chamado **Teste de Aceitação**. Assim, transformamos o teste de sistema em uma ferramenta de especificação e não apenas de validação.

Não se deve esquecer a necessidade de documentação do processo de negócio, essencial para o entendimento do ambiente em que o software está inserido. De maneira alguma um documento escrito torna-se obsoleto com o modelo apresentado acima, mas pode-se concluir que é possível diminuir em muito a documentação gerada que necessite uma manutenção complexa para manter o alinhamento com o código.

TDD x Matriz de rastreabilidade

Segundo Clayton Vieira Fraga Filho:

"A matriz de rastreabilidade oferece como grande facilitador a visualização global dos Casos de Uso e requisitos do sistema em uma tabela de forma gráfica, dando suporte ao analista para tomar decisões e descobrir problemas e sua solução de forma mais rápida, muitas vezes antes da fase de implementação."

Bastante utilizado nos modelos CMMI e MoProSoft, a Matriz de Rastreabilidade pode servir de ferramenta na estimativa do impacto de uma mudança nos requisitos de sistema. Entretanto, com a progressão do desenvolvimento, torna-se muito cara a manutenção dessa tabela tendo em vista sua característica não acoplada ao código. O nível de detalhamento possui papel importante no custo da manutenção.

A vantagem da utilização do Desenvolvimento Orientado a Testes está, em grande parte, à aderência ao código fonte e à automatização que os testes trazem. Mesmo não apresentando a visibilidade completa do impacto, já que testes podem falhar após uma alteração não prevista, existem maneiras de simular o uso de uma matriz de rastreabilidade, com o auxílio de ferramentas de desenvolvimento. IDEs mais completas possuem formas de verificar utilização de métodos no código fonte. Com a garantia da componentização e baixo acoplamento que o TDD traz, informações consistentes podem ser geradas, simulando assim uma matriz de rastreabilidade.

Conclusões

Durante a apresentação do conteúdo acima, tentei apresentar o conceito do Desenvolvimento Orientado a Testes de maneira a mostrar as vantagens de sua utilização. Apresentaremos nas próximas reportagens o lado prático de sua aplicabilidade. Fiquem atentos!

Referências:

Tudo Sobre TDD - Coding Dojo Floripa

<http://dojofloripa.wordpress.com/2007/09/10/tudo-sobre-tdd/>

A Maldita Comédia

<http://malditacomedia.blogspot.com>



**Victor Hugo
Germano**
(*Autor)

Bacharel em ciência da computação pela Universidade federal de Santa Catarina, especializado em Gestão estratégica de TI. Atualmente trabalhando na empresa Audaces auxiliando no processo de adoção de metodologias Ágeis.

Humor de Projetos

Quem disse que não existe
comédia em projetos?



LUMópolis

UML e Cotidiano de Projetos de forma divertida



Um cardápio de metodologias ágeis



Por Fábio Câmara

Um projeto de software está com sérios problemas e quando acordei percebi que não era um pesadelo. Peguei um livro e li: O mau gerenciamento pode incrementar os custos de um projeto de software mais rapidamente que qualquer outro fator, escreveu Barry Boehm em 1981. Temi pelo meu emprego e pensei: _ antes de desistir vou contratar muitos desenvolvedores. Peguei outro livro e li: Adicionar pessoas faz um atrasado projeto atrasar ainda mais, escreveu Frederick Brooks em 1995.

Neste cenário assustador questionei-me: o que vou fazer diferente se preciso de resultados diferentes? Como compreender os verdadeiros obstáculos do gerenciamento de um projeto de software?

Procurei por muitos caminhos mas a resposta que mais gostei me foi ensinada pelos métodos ágeis. Entre muitos aprendizados que tive, li que o pensamento ágil reconhece 5 obstáculos no gerenciamento de projetos de software. São eles: pessoas, tempo, funcionalidades, orçamento e recursos (excluindo pessoas). Com isto em mente, coloquei em prática um pouco de 4 propostas ágeis que estudei.

Vamos conhecer um pouco destas propostas ágeis. São elas XP, MSF, SCRUM e FDD.

eXtreme Programming

Métodos ágeis são propostas incomuns de técnicas para projetos de software. XP, como o nome sugere, é algo um pouco mais radical. Propostas estranhas de técnicas esquisitas fazem os gerentes de projetos temerem utilizá-las em grandes projetos. Para atrapalhar ainda mais as poucas iniciativas de se quebrar paradigmas, o radicalismo de alguns pensadores de XP afastam a compreensão sobre as reais vantagens e os benefícios para o negócio que podemos alcançar com estes métodos.

Um cardápio de metodologias ágeis

A programação em pares, um dos mais diferentes e originais métodos do XP, traz para o gerente do projeto a necessidade de uma posição sem hipocrisia. Ou você ama, ou você odeia. Até hoje não encontrei um gestor “em cima do muro” sobre esta técnica.

Como exemplo, vamos citar Karl Wiegers, famoso autor de livros técnicos, que afirma em seus ensinamentos que a programação em pares como forma de inspecionar código não é efetiva na redução dos defeitos. Segundo o mesmo, entre 25 % e 35 % é o ganho atingido no aspecto “defect reduction”. Esta afirmativa é questionada por David Anderson, outro grandioso autor, que defende um número percentual em torno de 50%.

Já em minha própria vivência, as reuniões em pé (stand-up meeting) são fabulosas em quase todos os aspectos. Particularmente acredito no pensamento de John Kennedy, ex-presidente dos EUA, que dizia: quando se quer não fazer nada fingindo que está trabalhando, entre em uma reunião.

Listamos a seguir os principais métodos polêmicos do XP.

- **Pair Programming;**
- **Continuous Integration;**
- **Stand-Up Meeting;**
- **Collective Ownership (código fonte coletivo);**
- **Refactoring;**
- **On-Site Customer;**
- **Generalists.**

Na minha leitura, o XP é uma proposta muito completa de ciclo de vida de desenvolvimento de software, agradando ou não um gestor de projetos. Um dos principais fundadores do XP chama-se Kent Beck.

FDD – Feature Driven Development

Criado entre 1997 e 1999 em Cingapura por um time liderado pelo Jeff De Luca, é uma simples compilação de práticas estabelecidas nos últimos 30 anos. Um de seus maiores desenvolvedores, Peter Coad, definiu a idéia de Feature Definition e Feature List. Diversos renomados autores participaram da concepção das idéias do FDD. Dentre estes destacamos: Tom De Marco, Tim Lister, Jerry Weinberg e Frederic Brooks.

Em sua essência, FDD é mais um método de gerenciamento de software do que um ciclo de vida de desenvolvimento de software.

Resumidamente, FDD é dividido em 5 fases que explicam sua função. São elas

- **Shape Modeling** – é uma forma de questionar se todos comprehendem o que é para fazer, analisar requisitos não-funcionais e modelo de arquitetura;
- **Feature List** – É a representação do escopo listando a compreensão do que é para ser feito e os requerimentos a serem desenvolvidos;
- **Plan by subject area** – É a modularização da lista em conjuntos de funcionalidades relacionadas, permitindo o desenvolvimento de parte do sistema autonomamente;
- **Design by feature set** – É uma orientação que determina o desenvolvimento com base no domínio do problema. Sugere-se nesta fase uma modelagem profunda e detalhada em UML;
- **Build by Chief Programmer Work Package** – É o empacotamento de pequenas funcionalidades, uma redução evolutiva que nasce na fase 2 até a fase 4. Prioriza-se este pacote, codificando suas funcionalidades e criando unit tests.

FDD define também 4 camadas de arquitetura de software:

- **UI** – User Interface;
- **PD** – Problem Domain (lógica do negócio);
- **DM** – Data Management;
- **SI** – Systems Interfaces.

Notamos que o “core” de todas as fases e das camadas de arquitetura é a funcionalidade (feature). Cada funcionalidade é definida com uma fórmula simples, que permite ser repetível e confiável. A fórmula da funcionalidade tem a seguinte estrutura:

<action> <result> <object>

Exemplo:

<action> O valor total de vendas
<result> Faturamento bruto mensal
<object> Produtos vendidos no período

MSF – Microsoft Solutions Framework

Disseram-me que MSF só funciona em projetos com tecnologias da Microsoft. É impressionante a associação restritiva que naturalmente os desinformados impõem as coisas.

Também já ouvi que metodologias ágeis só são recomendadas para projetos pequenos. Pior que isso! Ouvi de uma diretora de TI de um banco: _ Hoje nós somos ágeis (referindo-se a ausência de gestão e de análise de requisitos de seu departamento), queremos ser mais organizados.

Estudando o assunto há 9 anos, eu sempre considerei o MSF uma proposta equilibrada, aonde seus 2 modelos e suas 3 disciplinas nunca se apresentaram como uma espécie de bíblia sagrada, determinística ao sucesso ou fracasso de seu projeto. Inclusive na versão 3.1 do MSF, antes da formalização das propostas ágeis no ano de 2001, tinha como conceito chave: _ Stay agile, expect changes.

O MSF 4.2, possui duas novas instâncias: MSF for Agile Software Development e MSF for CMMI Process Improvement. Podemos afirmar que o MSF Agile é um mix de posições equilibradas, pois defende um SDLC (Software Development Life Cycle) mais curto com iterações de no máximo 4 semanas, contudo preserva a importância dos papéis definidos previamente e abomina a linha “todo mundo pode fazer tudo no projeto”. Outro quesito de destaque nesta fusão são os testes unitários e a preocupação com a cobertura de 100% do código fonte.

Um tema muito forte dentro do MSF Agile é integração. Metodologias ágeis necessitam que os “stakeholders” estejam presentes o tempo todo durante o projeto. Para isso são constituídos cenários de tarefas de trabalho que englobam atividades planejadas para um desenvolvedor. Estes cenários fazem parte de uma determinada iteração do plano de iterações. Esta iteração agrupa os cenários de desenvolvedores e os cenários de testes (que são efetuados em seguida ao desenvolvimento). Desta forma controla-se a integração de um time com papéis diversos dentro do projeto (usuário, desenvolvedor e analista de testes).

Para o MSF, um projeto precisa dos seguintes papéis (entende-se papéis como responsabilidades que devem ser assumidas por algum membro da equipe):

- **Program Manager**
- **Product Manager**
- **User Experience**
- **Tester**
- **Developer**
- **Architect**
- **Release Manager**

É árduo afirmar que foi o fundador do MSF, porém o grande patrocinador sempre foi a própria Microsoft. Para não ficar em branco com nomes, Michael Cusomano, Steve McConnell destacam-se entre muitos outros como personagens da história do MSF.

SCRUM

SCRUM é um método de gerenciamento de software que pode ser usado com XP ou MSF. É baseado na teoria do controle empírico de processos e seus fundamentos são originados na indústria de manufatura japonesa.

Ciclo de vida empírico é um ciclo baseado na observação dos fatos. Muitos gerentes de projetos não gostam do método reativo sugerido pelo SCRUM e preferem trabalhar com um planejamento que na minha leitura é um trabalho especulativo, pois tenta prever uma seqüência de atividades lineares. Por mais que o cronograma tenha um valor que é o planejamento prévio, ou seja, pensar antes de sair fazendo, perde muito em tentar prever atividades distantes cheias de dependências predecessoras e não facilita o controle das mudanças de requisitos.

Segundo o SCRUM, o desenvolvimento deve ser trabalhado em 3 níveis: Sprint, Release e Product.

O ponto central é que os requisitos são convertidos em uma lista que contém valores do cliente chamada Product Backlog. Um sub-conjunto desta lista é criado e chamado de Release Backlog. Este sub-conjunto é particionado mais uma vez transformando-se em Sprint, uma espécie de acordo de desenvolvimento de funcionalidades que após aceito pela equipe não deve ser mais alterado.

Praticando SCRUM, o que mais chama a atenção é a simplicidade. Controlar projetos desta forma é participar de um jogo competitivo e saudável em que todos se auto-avaliam todos os dias (daily stand-up meeting) tornando possível resultados e técnicas de melhoria contínua.

Um cardápio de metodologias ágeis

O gerente de projetos como conhecemos hoje, na proposta SCRUM, é chamado de SCRUM Master. Suas principais responsabilidades resumem-se em duas: Proporcionar a passagem técnica e retirar todos os impedimentos. A equipe do projeto é dividida em apenas 3 papéis: o SCRUM Master (coach), o Product Owner e a equipe.

Em diversos aspectos, as técnicas do SCRUM diferenciam-se das propostas convencionais. Destaque para:

- **Product Backlog;**
- **The 30-Day Sprint;**
- **The SCRUM Meeting;**
- **Team Size;**
- **The Sprint Review.**

Para saber mais sobre SCRUM, é imprescindível ler algum dos títulos lançados por Ken Schwaber.

O que experimentar

Independente de sua curiosidade para novos sabores, os métodos ágeis são uma honesta resposta a pergunta: o que vou fazer diferente se preciso de resultados diferentes?

Baseados em processos iterativos incrementais e empíricos, os métodos ágeis são indiscutivelmente diferentes das propostas tradicionais que são fundamentadas em processos cascata. Esta diferença pode ser a resposta a sua dúvida também.

Meus resultados mudaram significativamente quando entendi a diferença essencial da proposta ágil em comparação ao modelo que eu havia aprendido nos ensinamentos do PMI – Project Management Institute.

A orientação PMI é:



A orientação ágil é:



Considere na imagem que a seta vermelha é uma constante, a seta verde é mais ou menos variável e a seta lilás é a mais variável.

Compreendendo a diferença entre as duas e alinhando com as características das expectativas de seus “stakeholders”, você conseguirá alcançar resultados diferentes. Experimente!



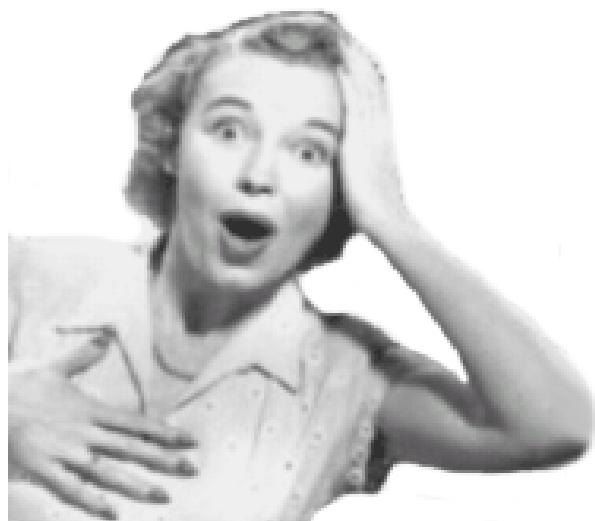
Fabio Câmara
(*Autor)

(fabio.camara@fcamara.com.br) é MCT, MCP, MCSA, MCAD Charter, MCITP, MCTS, MCSD.NET, MSF Practitioner, Certified ITIL Foundations e Certified SCRUM Master– Acredita em bons resultados em projetos com técnicas ágeis, principalmente para as características do mercado brasileiro.

Referências

Ágeis

Veja aqui, alguns sites e blogs indispensáveis para o mundo ágil



José Papo Weblog

Artigos de José Papo sobre engenharia e desenvolvimento de software, Ferramentas Open Source, programação e processos de desenvolvimento de software!!!

Desenvolva visualmente
Sistemas para web sem usar nenhuma linguagem de programação.

UML-Orientação a Objetos
Consultoria e Treinamento com Enterprise Architect

Anúncios Google

amazon.com Microsoft Office Project... Carl Chatfield, Ti... Head First PMP Andrew Stellman, J... New \$32.99

Privacy Info

SEXTA-FEIRA, OUTUBRO 26, 2007

MaintainJ : Engenharia reversa simples e ágil!

MaintainJ é um plugin para o Eclipse que gera **diagramas de sequência** e **diagramas de classe** com base na **engenharia reversa**.

Qual a sua diferença para outras opções no mercado? A maior de todas é que ela é uma das raras que gera o diagrama de sequência à partir de um cenário de uso da aplicação. De acordo com o RUP seria um diagrama de sequência de um dos fluxos da realização do caso de uso.

Esse processo é feito usando o conceito de engenharia reversa dinâmica. No mercado o mais comum é encontrarmos a **engenharia reversa estática**. Esta simplesmente lê o código e gera os elementos da UML (normalmente apenas as classes) respetivamente já a

Concluído

Site do papa **Martin Fowler**, quem é ele? Imagine um cara que foi citado em todos os principais livros sobre programação, modelagem, arquiteturas, padrões de projetos e refatoração de código. O resto, descubra por si só.

URL: martinfowler.com

Blog do **José Papo**, um ótimo lugar para você ter acesso a um material em português bem variado, profundo, feito com responsabilidade, e abordando temas como agile, OpenUP, UML, Modelagem e arquitetura.

URL: josepaulopapo.blogspot.com

Martin Fowler

Home Blog Articles Books About Me Contact Me ThoughtWorks

Martin Fowler

I am an author, speaker, and consultant on the design of enterprise software. On this site I keep as much information as I can on-line. There are links to my books, various on-line articles, and links to areas relevant to my work. My primary areas of involvement are in object-oriented development, refactoring, patterns, agile methods, enterprise application architecture, domain modeling, and extreme programming. I work for ThoughtWorks, an outstanding application development and consulting company.

Recent Updates

15 Oct 07: [Modifiability. Or is there Design in Agility](#)
The organizers of QCon London, earlier this year, asked me to do a conference session on modifiability of architecture. I thought that rather than listening to me, the audience might prefer listen to some of the ThoughtWorks architects whose ideas I usually repackage: Dave Farley, Ian Cartwright, Fred George, Erik Doernenberg, and Dan North. InfoQ has now put up a video of session.

26 Aug 07: [Language-Oriented Programming talk with Neal Ford](#)

Upcoming Talks
October
21-25 OOPSLA: Montreal
November
7-9 QCon: San Francisco, CA
More Details



Desenvolver ou não Desenvolver, eis a questão.

**Uma breve crônica
sobre a mecânica de um projeto de
desenvolvimento de software**

Por **Manoel Pimentel Medeiros**

Sei que você pode estar pensando, “**Desenvolver ou não desenvolver?** Ué, eu já ouvi uma frase parecida...”, exatamente, usei essa adaptação da célebre frase do livro Hamlet, escrita por **William Shakespeare**, para tentar mostrar uma breve metáfora para explicar o cotidiano de um projeto de desenvolvimento de software. Então, a idéia é levá-lo a uma reflexão sobre esse assunto, pois, que tal comparar a atividade de desenvolvimento de software, com uma peça de um grupo teatral, pois acompanhe comigo:

- O **Autor** da peça normalmente a escreve em conjunto de outros autores ou com **ajuda de revisores**.
- O **Diretor** e o produtor da peça precisam constantemente ler e reler os atos e cenas a fim de revisar a obra e buscar as melhores formas de montá-la.
- Os **figurinistas, iluminadores e cenógrafos** terão que conhecer muito bem o texto, os personagens e o ambiente de cada cena.
- Os **atores**, precisam ler e rê-ler o texto de forma minuciosa, afim de aprendê-lo, e principalmente precisam **entender** as “sub-linhas” do mesmo, para que possam compor melhores personagens

Desenvolver ou não Desenvolver, eis a questão.

- Cada ator, apesar de suas falas específicas, precisam **conhecer o texto como um todo**, para que haja sincronismo nas cenas e nas falas, para isso, então, cada ator, ensaiá o texto sozinho, lendo suas falas e dos outros personagens também, dessa forma, cada ator, consegue ter uma compreensão impar acerca da peça, e cada um poderá sugerir melhorias em suas cenas.
- Quando os atores estão ensaiando em grupo, sua atuação é posta ao conhecimento de outros atores e também do diretor, gerando então uma grande **margem para as sugestões, críticas, elogios e trocas de conhecimento** acerca das técnicas individuais de atuação.
- Dessa forma, como os atores adquirem um conhecimento pleno acerca da peça, é possível que quando necessário, **um ator, substitua outro ator**, na interpretação de um personagem em algum momento da peça, o nível de dependência individual é pequeno.
- E quando o grupo é pequeno, é comum, que haja uma “**polivalência**” entre seus membros, dessa forma, muitas vezes, uma mesma pessoa, é ator, maquiador, ajuda na direção e na produção da peça.
- É importante lembrar, que uma peça é ensaiada durante vários meses, e a cada ensaio, é detectado nas **necessidades de ajustes** em cada detalhe da peça, ou seja, cria como houvesse pequenos ciclos também conhecidos de iterações, para que a peça possa ser construída e melhorada a cada ensaio.

- E quando finalmente a peça é mostrada ao público, cada apresentação ajuda a gerar uma **retrospectiva** sobre o que pode ser melhorado para as próximas apresentações. Dessa forma, a peça passa por um processo de melhoria contínua para alcançar o máximo da satisfação do público.

Para terminar, veja que esses fatos e outros que não mencionei aqui, podem muito bem servir de metáfora ao processo de criação de um software, principalmente pela idéia central de que: mesmo que haja um esforço individual e solitário em alguns momentos, pela própria dinâmica da peça, **o resultado desse esforço é compartilhado e validado de maneira gradual** com todos os participantes, gerando um resultado final chamado obra coletiva, que será avaliado por um grande público, ou seja um alto nível de exigência de qualidade e encantamento, portanto meu caro amigo, pense nisso, e me diga se a nossa área de desenvolvimento de software não é semelhante à construção de uma peça teatral. Muito Obrigado e até a próxima.



**Manoel Pimentel
Meireiros**
(*Autor)

É Engenheiro de Software e CSM, trabalha com projetos Java pela Rhealeza Informática, Também é Dir. Editorial da Revista Visão Ágil, Colunista da Java Magazine, Membro do Desenvolvimento do NetBeans, Líder do projeto BoxSQL, e fundador do XPNorte, NUG-BR, e PróPatterns e Frequentemente palestra em eventos sobre processos e tecnologias. Maiores informações em: <http://manoelpimentel.blogspot.com>.

Tabuleiro de projetos



Uma breve crônica sobre estratégias em projeto de software, através de peças de xadrez.

Por **Manoel Pimentel Medeiros**

Estratégia, esse foi um dos pensamentos mais importantes para nossa evolução, porém, sempre imaginamos que o pensamento estratégico está distante da nossa realidade, apesar de que principalmente na área de desenvolvimento de software, o pensamento e ações estratégicas são fundamentais para o sucesso de um projeto. Durante muito tempo, devido a algumas visões equivocadas de alguns modelos de produção, fomos condicionados a pensar que devíamos apenas nos importar em apertar parafusos, não sabendo exatamente qual o parafuso certo e para que serviria aquele parafuso.

Porém, com o passar dos tempos, o poder do pensamento mais analítico foi tomando importância, e o poder do pensamento estratégico foi se tornando um fator determinante para o sucesso de muitas companhias.

Na verdade o pensamento estratégico, não é algo propriamente novo, ele está presente na história da humanidade há muito tempo, desde as antigas guerras que contribuíram sistematicamente para as mudanças de nossa civilização.

Você deve estar se perguntando, o que isso tem a ver com projetos de software? Eu lhe respondo: elementar meu caro leitor, hoje em dia o sucesso de muitos projetos de software está intimamente ligado ao uso de estratégias, porém, poucas as pessoas que participam de projetos, compartilham com essa visão.

Dado a importância que o pensamento e ações estratégicas têm para um projeto de software, resolvi brincar um pouco com essa idéia, usando como metáfora um jogo de xadrez, onde através da figura de cada peça, você entenderá de uma forma bem divertida e dinâmica, como tratar cada elemento de um projeto de software.

Mais atenção, leia atentamente a tabela abaixo, e principalmente reflita bem, sobre as correlações entre as peças do jogo xadrez e os elementos comuns em um projeto de software que estou tentando mostrar.

Tabuleiro de projetos

Peça	Significado no projeto	Ponto Forte	Ponto Fraco
 Peões	Simplicidade	Devido a sua simplicidade de movimentos, muitas vezes, é ignorado pelo adversário, porém, um peão é capaz de deixar em xeque um rei adversário ou até mesmo se tornar uma rainha podendo determinar o sucesso de um jogo.	Muitas vezes, para defender alguma peça ou quando fazemos um movimento errado, os peões são os primeiros a serem eliminados, ou seja, é muito fácil perder totalmente o poder da simplicidade durante o jogo.
 Torres	Prazo	Quando tudo está bem no jogo, raramente iremos mexer com a torre, mas quando necessário e quando bem manipulado é capaz de criar um bom xeque-mate.	Lembrando que a torre só pode ser movimentada na horizontal e vertical, quantas casas forem necessárias, note que há um limite de movimento, pois às vezes, gostaríamos que ela se movimentasse na diagonal, mas isso não é possível, ou seja, temos limitações ao tentar mover de forma eficiente essa peça.
 Bispos	Recursos	Representando a sabedoria no jogo, pode facilmente transitar em qualquer parte, devido aos seus movimentos na diagonal e usado com sabedoria, pode fazer ou facilitar várias opções de xeque-mate.	Usado frequentemente para proteger peças como a rainha ou o rei, e por transitar facilmente pelo terreno adversário durante um ataque, é comum perder-los, por excesso de zelo ou por demasiada ousadia.
 Cavalos	Criatividade	Pelo poder de alcance e a limitação de só se mover em forma de "L", força o jogador usa-lo com muita criatividade, dessa forma, somos capazes de iniciar um jogo ou fazer o xeque-mate usando o cavalo.	Frequentemente, o cavalo é usado para proteger a torre, o bispo, a rainha ou até mesmo o rei, dessa forma, é comum terminarmos o jogo, sem nenhum cavalo.
 Rei	O Software	É o objetivo do jogo, tanto para o ataque quanto para defesa, a vitória é quando atingimos o rei adversário, ou seja, o famoso xeque-mate.	Infelizmente, simplesmente quando nosso rei sofre um xeque-mate, perdemos o jogo (acho que deu para ser claro, certo?).
 Rainha	Qualidade	Esta é peça mais poderosa do jogo, pois ela pode se mover em qualquer direção, quantas casas forem necessárias, podendo ser fulminante quando usado para o ataque, garantindo uma vitória certa no jogo.	Porém, frequentemente, sacrificamos a rainha, para defender o rei, e quando isso acontece, nosso poder de vitória diminui drasticamente.

Como você deve ter percebido, é muito importante, que os elementos chaves de projetos sejam gerenciados com muita estratégia, a fim de maximizar as chances de sucesso, evitando dessa forma os famosos atrasos, estouros de orçamentos, software fora do escopo, enfim vários outros problemas que você deve conhecer.

Claro que não existe uma combinação mágica, que funcione como uma “bala de prata” para garantir o sucesso de um projeto, portanto, sempre que possível, lembre sempre desse esquema, afim visualizar o efeito de cada “movimento” seu em um projeto, ou seja, o que será ganho e poderá ser perdido em cada movimento no projeto, dessa forma, você poderá tomar as decisões de maneira mais conscientes e acertadas.

Espero que você tenha gostado desse texto, e principalmente tenha absorvido a idéia principal relacionada aos elementos de um projeto e sua relação com o jogo de xadrez, portanto, **muito obrigado e até a próxima**.



Apoios

aXmagnO

www.axmagnO.com



formação técnica **especialista**

> MSF + Agile Methods
> Ciclo de Vida de Desenvolvimento de Software
> VSTS para gerente de projetos

Conheça nosso cardápio para sua **formação.**



você pode,
se souber...

<http://www.fcamara.com.br>

F|camara
FORMAÇÃO | CONSULTORIA



www.tc4digital.com

