Falhe cedo, aprenda rápido: porquê utilizar métodos ágeis para criar excelentes produtos

Fail soon, learn fast: why use agile methods to create excellent products.

Igor Maciel Macaúbas[1]

2010

[1] Bacharelando em Sistemas de Informações - FIR - Faculdade Integrada do Recife igor@macaubas.com - http://macaubas.com

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Resumo

Criar produtos de sucesso não é um trabalho fácil, principalmente quando se fala em software. O mercado está cada vez mais agressivo, e é extremamente difícil conciliar as necessidades de inovação, criação e excelência dentro de orçamentos e prazos apertados que regulam a indústria de desenvolvimento. Neste mundo cada vez mais intolerante à falhas, é contra-intuitivo afirmar que falhar, e saber lidar com as falhas, pode muitas vezes ser uma vantagem competitiva imensa.

Palavras chave: desenvolvimento, metodologias, inovação, gestão, agile, waterfall, comparação

Abstract

Create successful products is not an easy job, and even tougher when we talk about successful software. The market is more aggressive every day, it's very hard to conciliate the needs of innovation and creativity with the tight schedules and budgets that rules the development industry. In a world where failure is not an option, it's counter-intuitive to say that failure, and knowing how to handle it, creates indeed a huge competitive advantage.

Keywords: software, development, methodology innovation, management, agile, waterfall, comparison

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Sumário

Introdução 4

O modelo Waterfall 5

Abordagem ágil 6

Agile v.s. Waterfall 8

Falhando cedo, aprendendo rápido 10

Referências Bibliográficas 11

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Introdução

Logo após divulgar a invenção da lâmpada elétrica, Thomas Edison foi questionado por um repórter sobre como ele conseguiu ser perseverante e continuar tentando criar o filamento elétrico, mesmo depois de ter falhado mais de 10 mil vezes. A resposta de Thomas Edison foi: "*Eu não falhei, nem uma única vez. O que eu fiz foi descobrir mais de 10 mil formas diferentes de como não se fazer um filamento elétrico*".

É seguindo esta linha de pensamento que os métodos ágeis de desenvolvimento de software surgiram. No final dos anos 60 e início dos anos 70, empresas de todo o mundo começaram a sentir os primeiros efeitos da crise do software, que foi causada por conta do aumento da complexidade dos sistemas de informação, que por sua vez se tornaram mais complexos por conta do aumento do poder de processamento dos computadores. Edsger Dijkstra descreve muito bem a situação, no seu artigo "The Humble Programmer", de 1972:

"*A maior causa da crise do software é que as máquinas se tornaram muito mais poderosas, por várias ordens de magnitude! Indo direto ao ponto: quando não existiam máquinas, programação não era um problema; quando nós tínhamos alguns poucos e fracos computadores, programação era um problema pequeno, e agora que nós temos computadores gigantes, programa-los se tornou um problema igualmente gigante.*" (Tradução livre)

Enquanto isso, as disciplinas de gestão e desenvolvimento de software não evoluíram na mesma velocidade. A crise se manifestou, e ainda se manifesta até os dias de hoje. O instituto The Standish Group, gera um relatório anual, desde 2000, que reporta como estão os projetos de desenvolvimento de software no mundo. Em 2009, através do documento CHAOS Summary, foi possível observar que a situação dos projetos não mudou muito ao longo dos anos: continuamos a ter uma taxa média de 30% de projetos concluídos com sucesso (dentro do prazo, dentro do custo orçado) - **Fig. 1**.

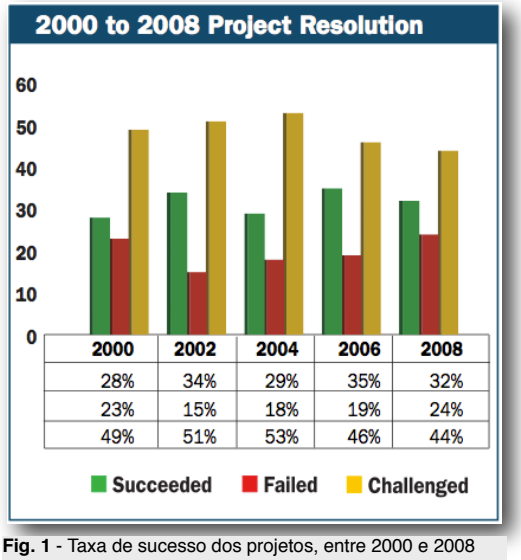

Fig. 1 - Taxa de sucesso dos projetos, entre 2000 e 2008 (*Fonte: CHAOS Summary 2009, The Standish Group International, Inc.*)

Além dos problemas de projeto, a crise do software também se reflete no produto final, que muitas vezes carregam os seguintes problemas:

- O software produzido é muito ineficiente;
- A qualidade do software é muito baixa (muitos bugs);
- O software produzido muitas vezes não atende os requisitos;
- O código é muito difícil de manter e evoluir.

Ao longo das últimas décadas, vários processos, modelos e metodologias foram criados, na tentativa de "domar" o desenvolvimento de software. Entretanto, não há "bala de prata" (do inglês, "*silver bullet*"), ou seja, não há uma abordagem definitiva que resolva todos os problemas criados por conta da grande complexidade de hardware e software.

Duas abordagem de desenvolvimento ganharam grande destaque nos últimos anos: O modelo de desenvolvimento *Waterfall* (em cascata) e o modelo, ou conjunto de métodos, ágeis de desenvolvimento de software. Em ambos os modelos, há um corpo de conhecimento para auxiliar na gestão e desenho (*design*) de projetos de software, corpo de conhecimento gerado com foco em resolver os problemas da crise do software. A maior dissonância entre os modelos está na filosofia por trás deles: enquanto o modelo *Waterfall* é centrado no processo (*process-centric*), o modelo ágil é centrado no humano (*human-centric*). O modelo ágil de desenvolvimento está em franca expansão na indústria de desenvolvimento de software, e segundo relatório do instituto *Forrester Research*, 35% das empresas entrevistadas aplicavam esse modelo no seu ciclo de desenvolvimento.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

O modelo *Waterfall*

O modelo *Waterfall*, ou cascata, utiliza uma abordagem determinística e seqüencial para desenvolver software, passando por fases bem distintas: análise, design, implementação, testes, integração e manutenção (**Fig. 2**). Essas fases variam de implementação para implementação do modelo, entretanto são basicamente estas. A saída de cada uma dessas fases serve de entrada para o início do trabalho das próximas fases. O modelo Waterfall foi definido por Winston Royce, em 1970, que usou como base um modelo mental de manufatura para concebe-lo.

Este modelo assume como premissa que desenvolvimento de software é um processo preditivo, linear, onde o cliente sabe o que quer, os desenvolvedores sabem como implementar, e nada (ou muito pouco) irá mudar durante o caminho.

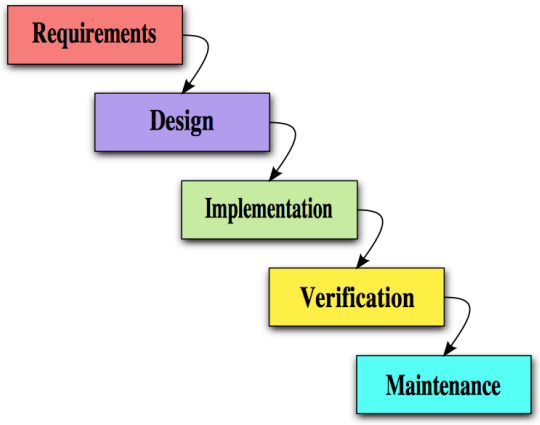

Fig. 2 - O modelo de desenvolvimento *Waterfall*

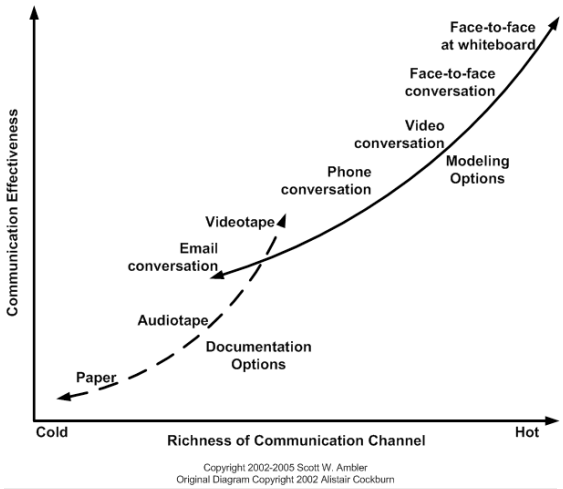

Fig. 3 - Efetividade da comunicação - Autor: *Alistair Cockburn*

Este modelo tem problemas graves, principalmente por condicionar um longo ciclo de *feedback* em termos do processo de desenvolvimento. O cliente só terá acesso ao produto final depois que todas as fases (ou pelo menos uma das fases) seja concluída, o que pode demorar meses. Ou pior ainda, muitas vezes os clientes iniciam projetos de desenvolvimento sem saber ao certo qual o problema que ele precisa resolver.

Um dos maiores argumentos de suporte desse modelo é que ele é fortemente orientado à documentação. Todas as fases geram vários artefatos, de diferentes tipos, que são a cada fase incrementados e passados para frente. Este argumento de suporte também se reverte ao mesmo tempo em uma das maiores críticas ao modelo: Alistair Cockburn, no seu livro "Agile Software Development", nos mostra que a comunicação escrita, não verbal, é o tipo de comunicação mais "fria" e menos efetiva existente - ou seja, quando dependemos somente da comunicação escrita para transmitir informações, é muito provável que informações se percam, ou sejam mal interpretadas do outro lado (**Fig. 3**).

Um outro forte argumento de suporte ao modelo *Waterfall* é que ele provê uma abordagem mais estruturada, menos caótica, ao processo de desenvolvimento de software. E, ao mesmo tempo em que oferece essa estrutura, ajuda a garantir a qualidade do produto final, investindo uma boa parcela de tempo do projeto a levantamento de requisitos e design do software. Steve McConnel nos mostra em seu livro "*Rapid Development: Taming Wild Software Schedules*" de 1996, que detectar e remover defeitos nas fases iniciais (de requisitos ou design) custam de 50 a 200 vezes menos do que corrigir o mesmo problema na fase de manutenção.

Entretanto, o próprio Steve McConnel, no seu livro "*Code Complete*", de 2004, faz duras críticas a esse argumento, pois dificilmente os requisitos e limitações são perfeitamente conhecidos nessas fases do projeto, tornando a conclusão dessas fases uma tarefa praticamente impossível. David Parnas, no seu livro "*A Rational Design Process: How and Why to Fake It*" diz:

"*Muitos dos detalhes [do sistema] se tornam conhecidos aos desenvolvedores à medida em que se avança na implementação [do sistema]. Algumas das coisas que nós aprendemos invalidam o design que pensamos no começo, e precisamos muitas vezes recomeçar do zero*".

Esse tipo de problema causa muitas vezes o cancelamento, ou grandes atrasos nos projetos. Em suma, o modelo proposto por Royce muitas vezes nos leva a cometer as mesmas falhas do passado, nos levando novamente a um momento de crise.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Aliado à isso está o fato de que muitas implementações do modelo *Waterfall* não valorizam o capital humano da forma que deveriam. O modelo mental para o trabalhador neste modelo é muitas vezes um modelo *Taylorista*, onde o trabalhador fará o que é comandado a fazer, sem se importar com a qualidade do seu trabalho ou em se exceder nas suas especialidades. Isto termina por causar problemas motivacionais, principalmente porque os trabalhadores envolvidos num projeto *Waterfall* não tem uma visão holística do projeto, e não conseguem ver o resultado do seu trabalho.

Abordagem ágil

A abordagem ágil, ou Agile, não é prescritiva nem determinística - ao invés disso, essa abordagem se utiliza de métodos empíricos de controle de processo para gerenciar o desenvolvimento de software. Também não é uma metodologia com fases bem definidas: no mundo ágil, existem diversos “modos” de trabalho, que oferecem mecanismos diferentes de controle do processo de desenvolvimento, mas compartilham da mesma filosofia por trás. O modelo ágil mais conhecido e praticado, segundo pesquisa da Forrester, é o Scrum: ele já está presente em 11% das empresas de desenvolvimento de software.

O movimento ágil surgiu oficialmente em 2001, com a criação do **Manifesto Ágil**, entretanto iniciativas isoladas já aconteciam desde a década de 90. O manifesto foi fruto de um trabalho conjunto de líderes e pensadores de desenvolvimento de software, que em Fevereiro de 2001 se reuniram para compartilhar histórias e experiências. Neste encontro, compararam projetos que falharam, projetos que foram bem sucedidos, e encontraram uma série de características comuns por trás dos projetos de sucesso. Com base nessas características, foi criado o **Manifesto Ágil para Desenvolvimento de Software (Fig. 4)**, que é baseado em 4 valores e 12 princípios:

Fig. 4 - O manifesto ágil - Fonte: http://agilemanifesto.org/iso/ptbr/

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Scrum

"Scrum" é um termos do esporte Rugby, e é uma formação onde todos os jogadores do time avançam em conjunto, como se fossem um só, passando a bola dentro da formação. O termo foi cunhado no artigo "The New New Product Development Game", de 1986, por Hirotaka Takeuchi e Ikujiro Nonaka. Neste artigo, o termo Scrum é utilizado pela primeira vez, com o sentido de ser um time multi-disciplinar e auto-contido. Jeff Sutherland e Ken Schwaber criam na década de 1990 um modelo de desenvolvimento de software baseado no artigo, e o batizam de Scrum.

O Scrum é um modelo ágil de desenvolvimento de software, que garante a entrega do produto em várias entregas iterativas e incrementais, em um curto período de tempo. Um dos pontos fortes por trás do Scrum é o seu foco no aprendizado ao longo do processo. Além disso, o Scrum também recomenda que os times sejam pequenos (5 - 9 pessoas), multidisciplinates (especialistas generalistas), e auto-contidos (que possam realizar 100% do trabalho sem dependências externas).

Desenvolvimento iterativo/incremental

Enquanto o modelo Waterfall encara desenvolvimento de software como uma atividade faseada e sequencial, o pensamento ágil encara desenvolvimento de software como um processo de aprendizado, onde o produto é incrementado a medida em que o time conhece mais sobre o produto a cada iteração. O modelo iterativo e incremental é muito bem exemplificado abaixo:

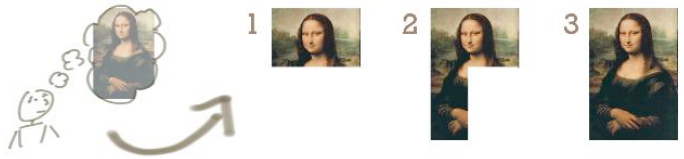

Fig. 5 - Desenvolvimento Incremental - Autor: *Jeff Patton*

Fig. 6 - Desenvolvimento Iterativo - Autor: *Jeff Patton*

O conceito ***Incremental*** consiste basicamente em ***adicionar*** coisas novas. Na **Figura 5** podemos ver um exemplo onde partes do quadro são incluídas incrementalmente.

O conceito ***Iterativo*** consiste em refinar o trabalho previamente feito, aperfeiçoando o resultado. O Scrum trabalha dessa forma: A cada iteração, as funcionalidades são melhoradas, e novo software é adicionado.

A grande vantagem de se trabalhar neste fluxo é o aprendizado. A cada iteração, o time pára para refletir sobre o trabalho realizado, e tem a oportunidade de implementar melhorias, tanto em termos de produto, como em termos de processo de desenvolvimento. Em termos de produto, o time melhora de acordo com o *feedback* recebido pelos usuários e clientes, e em termos de processo de desenvolvimento, o time aprende com seus erros e testa alternativas para resolver seus problemas.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Agile v.s. *Waterfall*

Comparações entre os dois modelos de trabalho são inevitáveis - afinal, antes de existirem os métodos ágeis de desenvolvimento, muito se fez no mundo do software seguindo métodos tradicionais de trabalho. Não que estes sejam ineficientes ou não sirvam: Agile é uma alternativa, que deve ser considerada, e funciona muito bem em muitos casos.

Waterfall

O modelo de desenvolvimento Waterfall assume as seguintes premissas:

1. O cliente sabe o que quer;
2. Os desenvolvedores sabem como construir o que o cliente quer;
3. Nada vai mudar ao longo do caminho.

Em um estudo realizado pela *N-Cycles Software Solutions* em 2002, foram destacadas as principais características de cada modelo.

Pontos fortes do modelo *Waterfall*:

A maioria dos pontos fortes do modelo *Waterfall* são conseqüência dos princípios de estrutura formal que existem por conta do uso deste modelo. São eles:

- Facilidade em analisar potenciais mudanças;
- Possibilidade de se coordenar grandes times, mesmo que distribuídos geograficamente;
- Possibilidade de controle orçamentário preciso;
- Menos tempo demandado de SMEs (*Subject Matter Experts* - Especialistas de negócio).

Devido à vasta documentação gerada em cada uma das fases do processo, é relativamente fácil e rápido analisar o impacto de mudanças de qualquer natureza, sejam elas de recurso, escopo ou tempo. Mas, para conseguir atingir esse grande nível de detalhe, muito tempo é demandado nas fases de coleta e análise de requisitos. Mesmo assim, uma das maiores justificativas em se empregar o modelo Waterfall é a baixa demanda de tempo de SMEs - nos modelos Waterfall, eles só são envolvidos na hora de coletar os requisitos do projeto, e não são envolvidos durante todo o restante do andamento.

Pontos fracos do modelo *Waterfall*:

Apesar da maioria dos seus pontos fortes serem conseqüência da estrutura formal requerida por este modelo de trabalho, suas principais fraquezas também são fruto dessa estrutura:

- Falta de flexibilidade;
- Dificuldade de prever todas as necessidades no início;
- Conhecimento tácito é perdido a cada mudança de fase;
- Falta de coesão de time;
- Falhas de design não são descobertas até a fase de Testes/Homologação.

Apesar de mudanças poderem ser melhor analisadas numa abordagem waterfall, o tempo requerido para implementar cada mudança pode ser significativo. Isso acontece por causa da natureza estruturada do modelo, e esses efeitos são mais agudos ainda quando mudanças são freqüentes ou grandes. Um dos maiores problemas é perda do conhecimento tácito a cada mudança de fase: remetendo novamente à **Fig. 3**, é fácil ver a causa dessa perda de conhecimento.

Agile

O modelo de desenvolvimento *Agile* assume as seguintes premissas:

1. O cliente descobre aos poucos as suas necessidades;
2. Os desenvolvedores descobrem como construir o que o cliente quer;
3. Muitas coisas vão mudar ao longo do caminho.

O modelo de desenvolvimento Ágil segue um paradigma iterativo e incremental. Apesar do foco do estudo da *N-Cycle Solutions* ser sobre o modelo de desenvolvimento Iterativo, o mesmo estudo se aplica nesta comparação, pois esta é uma característica determinante dos métodos ágeis.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Pontos fortes do modelo Iterativo:

- *Feedback* rápido dos usuários finais e clientes;
- Flexibilidade para lidar com a evolução dos requisitos;
- Falhas de design são descobertas rapidamente;
- Facilidade de liberar novas funcionalidades;
- Maior motivação e produtividade;
- Pouco ou nenhum conhecimento tácito perdido.

O *feedback* dos usuários finais são baseados em versões funcionais do produto, que ficam disponíveis logo no início do ciclo de vida do sistema. Como o time de desenvolvimento recebe *feedback* mais cedo do desenvolvimento, mudanças são incorporadas de forma muito mais suave no produto final. Além disso, a abordagem de time que é característica marcante dos modelos iterativos, aumentam a motivação e a produtividade dos envolvidos. A produtividade é aumentada por causa do senso de propriedade (ownership) que o time tem do resultado final.

Pontos fracos do modelo Iterativo:

- Dificuldade em coordenar times grandes;
- Podem resultar em projetos que nunca acabam, se não gerenciado corretamente;
- Tendência em não documentar coisas importantes;
- Dificuldade em prever quais funcionalidades serão implementadas em um orçamento/tempo determinado.

Apesar de um dos pontos fortes ser a velocidade para se adaptar a mudanças, é importante que os responsáveis pelo projeto saibam determinar quando as principais necessidades de negocio forem atingidas, para que o projeto não entre num ciclo de perpetuação. Um outro fator que aumenta os riscos de projetos Iterativos é que, como não há tempo determinado para documentação, o resultado pode ser um sistema difícil de manter e evoluir. Mas, para mitigar esse tipo de risco, existem práticas de engenharia que podem ser aplicadas, tais como *TDD* (Test-Driven Development) e *Refactoring*.

Conclusão: Agile vs Waterfall

Não há bala de prata. Um modelo de trabalho não irá garantir que o projeto seja bem sucedido: ele é apenas um meio para atingir um fim. Nenhum modelo de trabalho irá substituir experiência e conhecimento dos indivíduos, objetivos de negócio claramente definidos e um correto time-to-market. Qualquer ferramenta pode ser mal utilizada.

Enquanto o modelo Waterfall pode ter muito sucesso na construção de software onde os requisitos são bem conhecidos, e há necessidades de se distribuir o time geograficamente, o modelo Ágil terá mais sucesso em construir software novo, onde requisitos mudam e evoluem rapidamente.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Falhando cedo, aprendendo rápido

Uma das grandes vantagens do modelo de desenvolvimento iterativo é a velocidade do seu ciclo de *feedback*: as iterações normalmente duram entre 1 a 4 semanas, e ao final, já é possível receber *feedback*, do usuário final. Somente este fato já oferece um grande espaço para melhorias: dessa forma, é mais fácil detectar erros cedo, e corrigi-los cedo. Este é o princípio por trás do falhe cedo, aprenda rápido.

Desde que começaram a se utilizar de métodos ágeis, várias empresas vêm se aproveitando dessa característica inerente à esta forma de trabalho, e vêm tendo cada vez mais sucesso junto aos seus clientes. É o caso por exemplo da empresa norte-americana salesforce.com, que Steve Greene e Cris Fry apresentaram na conferência Agile 2007.

Depois de 7 anos de existência, a empresa entrou em uma espécie de paralisia causada pelos processos que foram criados para gerenciar e coordenar uma equipe de mais de 200 pessoas em pesquisa e desenvolvimento. Entregar novo software era cada vez mais difícil, e enquanto a quantidade de funcionalidades entregues por cada time caía, o número de dias entre os *releases* aumentava assustadoramente a cada ano. Um dos grandes vilões da história: *Feedback* tardio, só no final do ciclo de desenvolvimento, das funcionalidades que estavam sendo desenvolvidas.

Algo precisava ser feito, e a solução adotada pela salesforce.com foi transformar a empresa, adotando desenvolvimento ágil em todo o seu ciclo de desenvolvimento. O resultado imediato disso foi logo percebido: os times começaram a entregar mais, e o número de dias entre os *releases* começou a cair. A satisfação dos clientes aumentou, pois eles começaram a ter suas opiniões consideradas, e os times de desenvolvimento ficaram mais felizes com o resultado do seu trabalho. Nesta apresentação, Steve Greene dá algumas dicas de como ter sucesso com Agile, e uma delas diz: “Experimente, seja paciente, e espere cometer erros”.

Este modo de pensar traz enormes benefícios, principalmente em pequenas empresas, que estão desenvolvendo produtos novos, onde a capacidade de receber *feedback* rapidamente dos seus usuários pode significar a diferença entre a vida e a morte. Foi assim com o Flickr, popular serviço de hospedagem de fotos e vídeos na internet. O Flickr emergiu como um conjunto de ferramentas, que faziam parte de um jogo online, no estilo MMORPG (Massive, Multiplayer, Online Role Playing Game). Entretanto, o seu foco de negócio mudou completamente baseado no retorno obtido dos seus usuários. Em um certo momento de 2006, o Flickr divulgou que o seu *release* era uma versão *Gamma* - indicando que o software que estava no ar estava sendo pesadamente testado e modificado baseado nas opiniões dos seus usuários. Um produto que tinha tudo para dar errado no começo, fomentou uma revolução, e hoje é o 31o. primeiro site mais acessado da Internet mundial, de acordo com o ranking da empresa Alexia, que mede a audiência de portais na internet.

Cada vez mais empresas, principalmente novas e pequenas empresas, têm procurado no Agile uma forma de ter sucesso nas suas empreitadas. Até os gigantes da internet, como o Google, têm inovado na velocidade e na forma com que liberam novas funcionalidades para os seus usuários - colhendo *feedback* constante, e respondendo rapidamente ao mercado.

Um exemplo recente dessa velocidade de resposta foi um experimento que o Google fez na sua página inicial. O Google liberou uma funcionalidade que permitia que os usuários registrados com Google Accounts personalizassem o fundo de tela da página principal do Google (www.google.com). Eles haviam liberado esta funcionalidade por 24 horas, para testar a aceitação dos usuários, mas devido ao enorme *feedback* negativo que receberam, terminaram removendo a mesma antes mesmo do fim do prazo estipulado. É este tipo de ciclo de *feedback* que é possibilitado por métodos ágeis, e é este tipo de aprendizado que é potencializado devido à velocidade do *feedback* recebido. E isso, hoje em dia, significa a diferença entre o sucesso e o fracasso do seu produto. Não evite falhas: aprenda com elas.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Referências Bibliográficas

DIJKSTRA, E.; EWD 340: The Humble Programmer. Commun. ACM 15 (1972), 10: 859–866.

ROYCE, W.; Managing the Development of Large Software Systems, Proceedings of IEEE WESCON 26 (August): 1–9.

MCCONNELL, S.; Rapid Development: Taming Wild Software Schedules. 1996, Microsoft Press. ISBN 1-55615-900-5.

MCCONNELL, S.; Code Complete, 2nd edition. 2004, Microsoft Press. ISBN 1-55615-484-4.

MCCONNELL, S.; Software Estimation: Demystifying the Black Art. 2006, Microsoft Press. ISBN 0-7356-0535-1.

PATTON, J.; I don't know what I want. Disponível em: <http://www.agileproductdesign.com/blog/dont_know_what_i_want.html>. Acesso em: 27/06/2010.

BECK, K., BEEDLE, M., et. al; Manifesto for Agile Software Development. Disponível em: <http://www.agilemanifesto.org/>. Acesso em: 27/06/2010.

NARAYAM, S.; Fail Fast. Disponível em: <http://blog.sriramnarayan.com/2010/02/fail-fast.html>. Acesso em: 27/06/2010.

CRAIG, K.; Birthplace of Agile?. Disponível em: <http://marcatopartners.com/2010/03/11/birthplace-of-agile/>. Acesso em: 27/06/2010.

PROULX, M.: Forrester reports: Agile Development Mainstream Adoption Has Changed Agility. Disponível em: <http://analytical-mind.com/2010/02/08/forrester-reports-agile-development-mainstream-adoption-has-changed-agility/>. Acesso em: 27/06/2010.

THOMSON, I.; User feedback kills Google picture experiment. Disponível em: <http://www.v3.co.uk/v3/news/2264588/google-kills-picture-experiment>. Acesso em: 04/07/2010.

WILLKINSON, C.; The Iterative Web App: A new compose interface for Gmail on iPad. Disponível em: <http://googlemobile.blogspot.com/2010/06/iterative-web-app-new-compose-interface.html>. Acesso em: 04/07/2010.

GREENE, S., FRY, C.; Large Scale Agile Transformation: How salesforce.com revolutionized their R&D development methodology in a Big Bang way. Disponível em: <http://www.slideshare.net/sgreene/salesforcecom-agile-transformation-agile-2007-conference>. Acesso em: 04/07/2010.

MARKS, D.; Development methodologies compared: why different projects require different development methodologies. Disponível em: <http://www.ncycles.com/e_whi_Methodologies.htm>. Acesso em: 28/06/2010.

COCKBURN, A.; Communication on Agile Software Projects. Disponível em: <http://www.agilemodeling.com/essays/communication.htm>. Acesso em: 28/06/2010.

GREENE, S.; Weird myths we believe in business. Disponível em: <http://www.slideshare.net/sgreene/weird-myths-in-business>. Acesso em: 04/07/2010.

THE STANDISH GROUP INTERNATIONAL; CHAOS Summary 2009. Disponível em: <http://standishgroup.com/newsroom/chaos_2009.php>. Acesso em: 02/07/2010.

WIKIPEDIA, et. al; Software Crisis. Disponível em: <http://en.wikipedia.org/wiki/Software_crisis>. Acesso em: 27/06/2010.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

WIKIPEDIA, et. al; Flickr. Disponível em: <http://en.wikipedia.org/wiki/Flickr>. Acesso em: 02/07/2010.

WIKIPEDIA, et. al; Waterfall Model. Disponível em: <http://en.wikipedia.org/wiki/Waterfall_model>. Acesso em: 02/07/2010.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.

Este artigo está disponível na sua íntegra, em formato PDF, no endereço: http://macaubas.com

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/br/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, CA, 94105, USA.