SOFTWARE DE QUALIDADE: METODOLOGIAS E DESAFIOS

Daniel Ferreira Ramos
Lucas de Araújo Coelho
Joseneuza Julita Pimenta de Aguiar
Rogério Oliveira da Silva

Resumo: A busca por qualidade nos processos de software deve-se tornar uma realidade. Este artigo, busca e afirma, antes de qualquer coisa, a necessidade de se documentar e aplicar os mais altos recursos e metodologias de qualidade no método de construção de software. Compondo o problema, os negócios atuais exigem o aumento de produtividade e melhor qualidade, com rápido desenvolvimento e distribuição. Desafios mostram-se presentes durante toda a construção. Há vários problemas comuns no desenvolvimento de software. A busca das melhores práticas comerciais que atacam a raiz deste problema de desenvolvimento de software deve ser objetivo de todas as empresas.

Palavras chave: Qualidade. Processos. Metodologias. Software.

Abstract: *The quest for quality in the processes of software must become a reality. This article, search and claims, before than anything else, the need to document and apply the highest features and quality methodologies in software construction method. Compounding the problem, the current business require the increased productivity and better quality, with rapid development and distribution. Challenges are present throughout the construction. There are several common problems in software development. The search of the best commercial practices that attack the root of the problem for software development should be the target of all companies.*

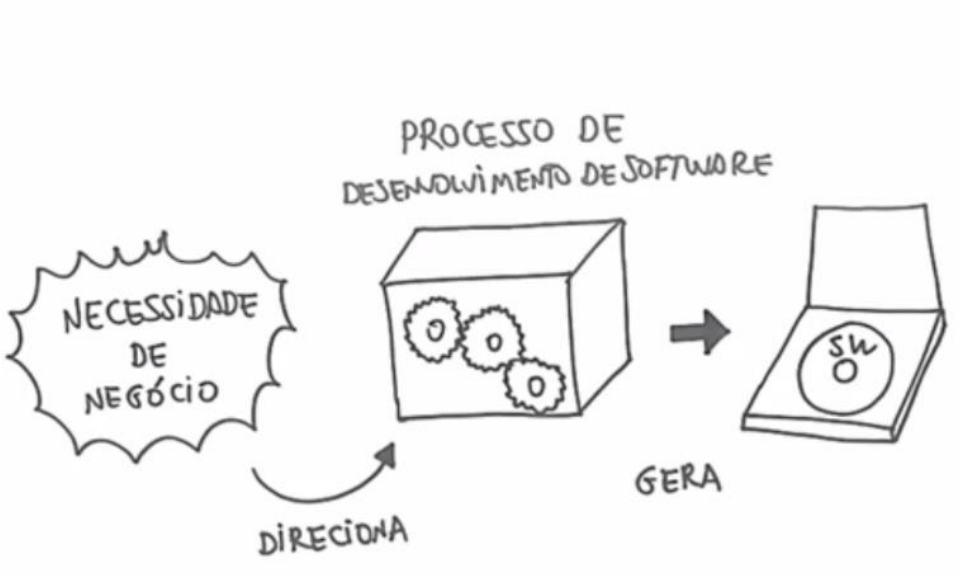
Keywords*: Quality. Processes. Methodologies. Software.*

1. Introdução

Muitas pessoas tomam o termo software como 'programas de computador'. De fato, embora não equivocada, esta é uma visão muito restrita. De acordo com Pressman (2011), softwares não são apenas os programas, mas também toda a documentação associada a toda uma configuração; a qual é necessária para fazer com que estes programas sejam executados corretamente. Ainda de acordo com Pressman (2011), um sistema de software usualmente, consiste em um número de programas separados, arquivos de configuração, documentação do sistema e de usuário, com um ou vários objetivos previamente estabelecidos durante o levantamento de requisitos. Com este foco, o uso de métodos sistemáticos e de uma abordagem organizada são as melhores formas para a obtenção de um software da mais alta qualidade. Logo, é valido questionar: O que é qualidade? Em uma busca no dicionário da língua portuguesa, tem-se definida a qualidade como: "propriedade, atributo ou condição das coisas ou das pessoas capaz de distingui-las das outras e de lhes determinar a natureza. Como um atributo de um item, a qualidade se refere a coisas que de 2 a 2 podem ser medidas, ou seja, comparadas com padrões conhecidos, tais como, tamanho, cor, propriedades elétricas, maleabilidade, etc." (FERREIRA, 2010). Entretanto, é mais difícil categorizarmos qualidade em software do que em objetos físicos, uma vez que software é uma entidade intelectual.

Observa-se então que qualidade, em linhas gerais, é um termo relativo e complexo: o que é valoroso e de qualidade para um consumidor, é extremamente inadequado para outro. Em um nível mais pragmático, Garvin (1988), da Harvard Business School, sugere que "qualidade é um conceito complexo e multifacetado", que oscila de acordo com a necessidade de cada negócio (Figura 1). No entanto, métricas de qualidade de software surgem desde a década de 70 e vêm se desenvolvendo de forma a ajudar no processo de desenvolvimento de software.

Figura 1 – Processo de desenvolvimento de software com qualidade



Fonte: https://youtu.be/n8sAGdxmsaQ (acessado em 03/09/2016)

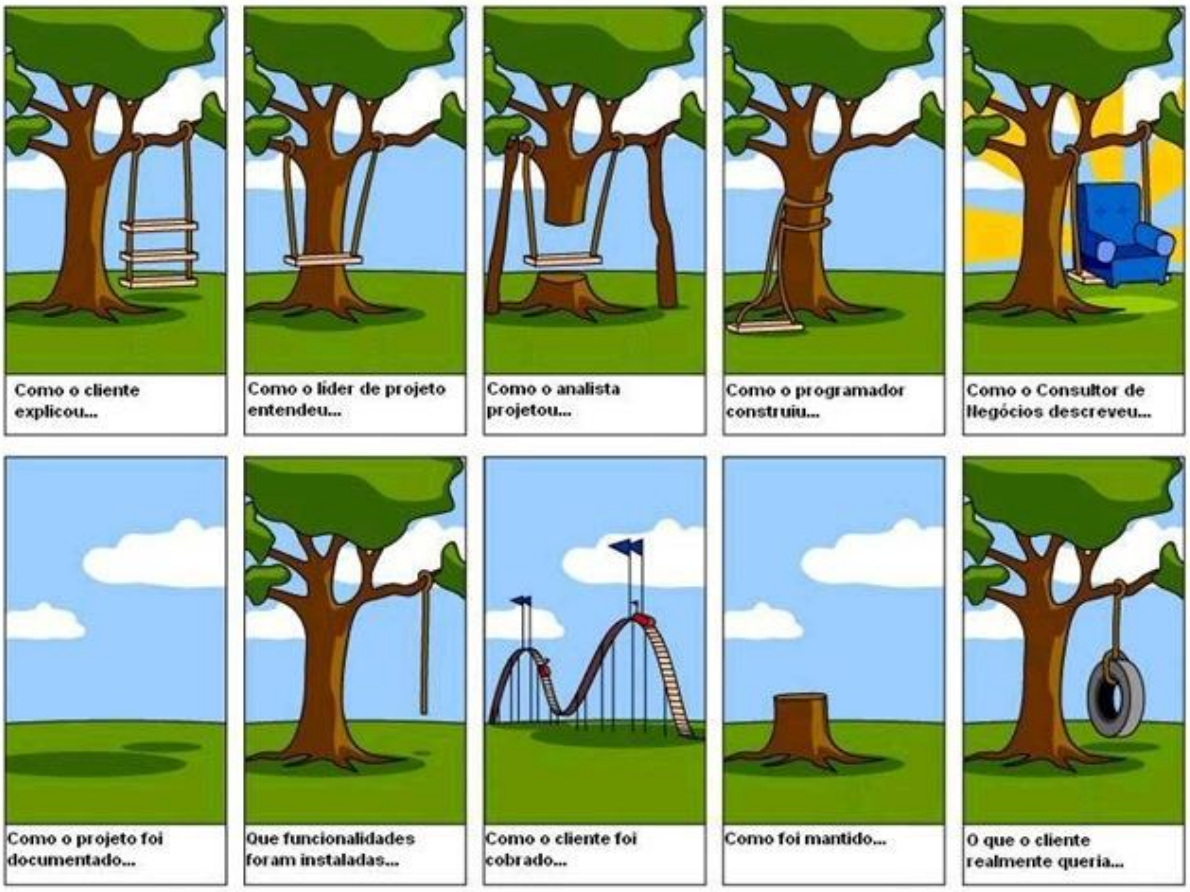
2. Contexto atual e histórico de Software e Qualidade

A busca por um produto de auto nível ou serviço de qualidade é o objetivo da maioria das organizações. Não é mais aceitável entregar software de baixa de qualidade e só então reparar problemas e deficiências depois da entrega ao cliente (Sommerville,2001). Neste sentido, software é equivalente a qualquer outro produto manufaturado: como carros, televisões ou computadores.

Historicamente, a noção de "qualidade de software tem sido a de que o produto deve ir de acordo com as suas especificações" (Crosby, 1979). No mundo ideal, esta definição deve-se aplicar a todos os produtos, mas para software as especificações devem ser orientadas conforme as características do produto que o cliente deseja. Contudo, a empresa desenvolvedora do software pode ter seus próprios requisitos (como requisitos de manutenção e construção) que não estão inclusos na especificação do projeto. Ainda de acordo com Sommerville (2001), ainda não se sabe como especificar certas qualidades de software de uma forma não ambígua e exata (como manutenibilidade e eficiência, por exemplo). O problema de não saber especificar corretamente o que o sistema deverá fazer é muito antigo. Pompilho (1995) cita um exemplo do relatório produzido por Mckinsey Global Institute, em 1968, e mencionado por B. Langefords e B. Sundgren (engenheiros do McKinsey), onde se afirmava que dois terços das empresas ali estudadas, estavam desapontadas com o atendimento recebido em sistemas de informação. Portanto, embora um software possa ir de acordo com suas especificações, usuários podem não considerá-lo de boa qualidade. Isso ocorre porque nem sempre o cliente/usuário consegue passar de maneira clara a sua real necessidade. Assim como também o analista de

requisitos nem sempre consegue compreender e/ou documentar com exatidão as expectativas do cliente em relação ao produto. Isso justifica o que já se sabe a respeito de erros em software: a maior parte dos erros encontrados estão nos requisitos, seguido pela modelagem (Figura 2).

Figura 2 – Charge (Tree Swing Cartoon) a respeito do levantamento de requisitos e comunicação



Fonte: http://www.projectcartoon.com/cartoon/3 (acessado em 15/09/2016)

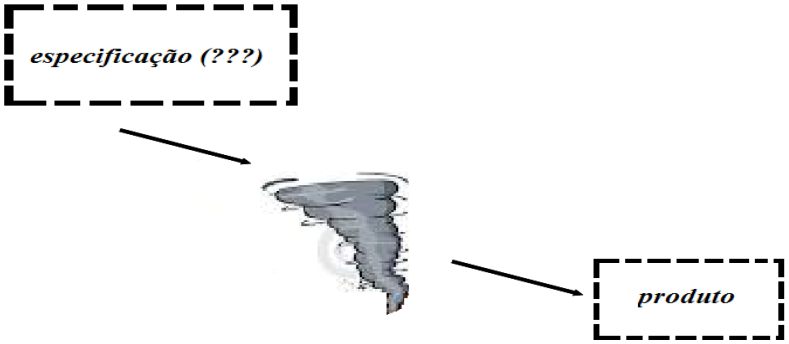
Segundo Tonsing (2008), são muitas as histórias de projetos de sistemas que, ao serem concluídos não apresentam os resultados esperados pelos usuários; muitas delas, infelizmente são recorrentes da ausência de feedback, em que o analista não utiliza o feedback como deveria e acaba tendo um entendimento diferente da ideia que o usuário gostaria de ter transmitido. O que torna importante por parte do analista uma checagem de seu entendimento quanto ao assunto.

3. Metodologias Tradicionais

Originalmente, metodologias de processo software foram propostos para trazer ordem ao caos existente na área de desenvolvimento de software. A história tem demonstrado que essas metodologias tradicionais proporcionaram uma considerável contribuição quanto à estrutura utilizável no trabalho de engenharia de software e forneceram um roteiro razoavelmente eficaz para as equipes de software (PRESSMAN, 2011).

Segundo Soares (2004) uma das metodologias tradicionais mais utilizadas até hoje é o modelo Clássico ou Cascata. Anteriormente, os softwares eram desenvolvidos pelo que pode ser chamado de "Codifica-remenda" (Figura 3).

Figura 3 – Exemplo de um ciclo de vida caótico



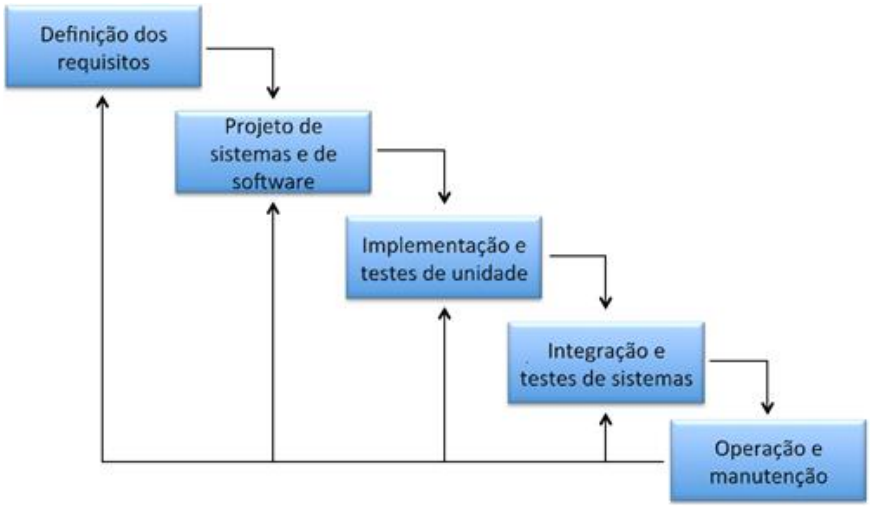
Fonte: Adaptado de Paula Filho (2005).

De acordo com Paula Filho (2005) esta metodologia inicia a partir de uma especificação (ou nem isto), os desenvolvedores começam imediatamente a codificar, remendando à medida em que os erros vão sendo descobertos. Nenhum processo definido é seguido. Para alguns desenvolvedores, esta metodologia é atraente porque não exige nenhuma sofisticação técnica ou gerencial. Por outro lado, é uma metodologia de alto risco, impossível de gerir e que não permite assumir compromissos confiáveis.

3.1. - Cascata

Uma das primeiras metodologias criadas para minimizar os problemas destacados acima foi a metodologia Cascata, representando um grande avanço no desenvolvimento de software. De acordo com Sommerville (2003) o primeiro modelo (‘cascata’) publicado do processo de desenvolvimento de software originou-se de outros processos de engenharia (Figura 4). (ROYCE, 1970 apud SOMMERVILLE, 2003, p. 37).

Figura 4 – Modelo cascata,



Fonte: Sommerville (2003)

Por causa da sequência em cascata de uma fase para outra, essa metodologia é conhecida como ‘modelo em cascata’. Os principais estágios do modelo retratam as atividades de desenvolvimento fundamentais:

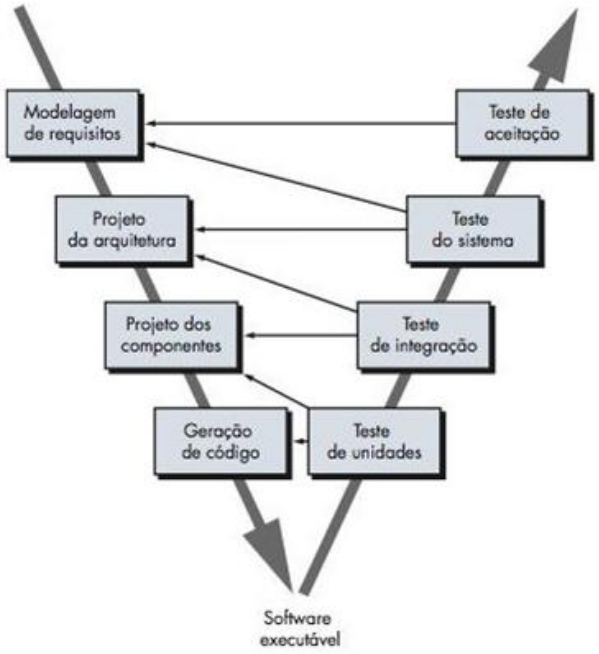
1. *Análise e definição de requisitos (especificação de requisitos):* As funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta aos usuários

do sistema. Em seguida, são definidos em detalhes e servem como uma especificação do sistema.

2. *Projeto de sistemas e de software:* O processo de Projeto de sistemas Agrupa os requisitos em sistemas de hardware ou de Software. Ele estabelece uma arquitetura do sistema geral. O projeto de software envolve a identificação e a descrição das abstrações fundamentais do sistema de software e suas relações.
3. *Implementação e teste de unidades:* Durante esse estágio, o projeto de software é compreendido como um conjunto de programas ou de unidades de programa. O teste de unidade envolve verificar que cada unidade atenda a sua especificação.
4. *Integração e teste de sistemas*: As unidades de programa ou programas individuais são integradas e testadas como um sistema completo a fim de garantir que os requisitos de *software* foram atendidos. Depois dos testes, o sistema de *software* é entregue ao cliente.
5. *Operação e Manutenção*: Normalmente, esta é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em operação, a manutenção envolve corrigir erros que não foram descobertos em estágios anteriores do ciclo de vida ou aumentar as funções desse sistema à medida que novos requisitos são descobertos.

De acordo com Pressman (2011) o modelo cascata possui uma variação denominada modelo V (Figura 5).

Figura 5 – Modelo V, variação do modelo cascata.



Fonte: Pressman (2011)

Segundo Pressman (2011) O modelo V (BUC, 1999 apud PRESSMAN, 2011, p. 60) descreve a relação entre ações de garantia da qualidade e as ações associadas à comunicação, modelagem e atividades de construção iniciais. A conexão entre o lado esquerdo e direito do modelo V implica que, caso sejam encontrados problemas durante a verificação e a validação, o lado esquerdo do V pode ser executado novamente para corrigir e melhorar os requisitos, o

projeto e a codificação, antes da execução das etapas de teste do lado direito do V, em outras palavras, o modelo V torna mais explicitas algumas iterações e repetições do trabalho, ocultas no modelo cascata (PFLEEGER, 2004).

De acordo com Pressman (2011) alguns dos problemas encontrados às vezes quando se aplica o modelo cascata, temos:

1. Os projetos de software reais construídos e evoluídos na indústria de software raramente seguem o fluxo sequencial que o modelo foi proposto. Ainda que esse modelo em forma sequencial possa conter iterações, podendo passar diversas vezes pelas mesmas atividades, ele o faz indiretamente. Então mudanças podem ocasionar confusões na medida em que a equipe de projeto prossegue.
2. É muito raro e difícil para o cliente saber e estabelecer claramente todas as suas necessidades. O modelo cascata é muito fortemente fundamentado nisso e tem dificuldade para adequar à incerteza natural que existem no início dos projetos.
3. O cliente precisa ter paciência. Uma versão operacional (pronta para ser executada no cliente) não estará disponível até estar próximo ao final do projeto. Se tiver um erro grave nas etapas iniciais, como uma especificação mal compreendida e mal especificada, poderá haver um resultado desastroso.

Outro grande problema citado por Pressman (2011) que tem com os projetos que usam metodologias cascata é a dependência criada, porque para que algumas equipes do projeto possam dar continuidade a sua tarefa é necessário aguardar que outras completem suas tarefas. O tempo gasto nessa espera pode exceder o tempo gasto em trabalho produtivo que levaria à conclusão do projeto.

4. Metodologias ágeis

Os conceitos existentes hoje relacionados ao desenvolvimento ágil de software nasceram em meados de 1990, motivados por uma reação adversa aos chamados "métodos pesados" de desenvolvimento de software, caracterizados por um formalismo muito grande nas documentações e regulamentações, sendo, na sua maioria, micro gerenciados pelo tradicional e burocrático modelo em cascata. (SBROCCO; MACEDO, 2012).

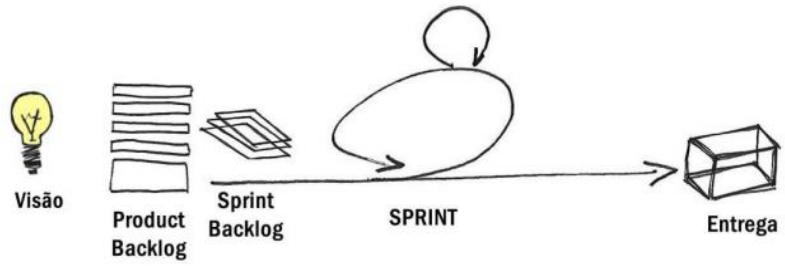
Com o objetivo de encontrar melhores maneiras de desenvolver softwares, dezessete renomados desenvolvedores assinaram, em 2001, o Manifesto para o Desenvolvimento Ágil de Software. Este documento contempla um conjunto de valores e princípios em que se valorizam mais os indivíduos e as interações do que os processos e as ferramentas, mais os softwares funcionais do que a documentação, mais a colaboração com o cliente do que a negociação de um contrato e mais as respostas às mudanças do que um plano formal (BIANCOLINO & TERLIZZI, 2014).

Com objetivo primordial ao desenvolvimento de código-fonte de qualidade que atenda às necessidades do cliente, as metodologias ágeis pretendem, entre outras coisas, assegurar que o cliente tire proveito da aplicação o quanto antes, fazendo com que receba constantemente partes do software na medida em que são concluídas. Existem diversas metodologias e frameworks ágeis, os mais comuns e utilizados são o Scrum, Extreme Programming(XP), Feature Driven Development(FDD), Dynamic Systems Development Method(DSDM), Crystal, Lean, KanBan, e outros. (RIBEIRO, 2015). As metodologias ágeis abordadas neste artigos serão o Extreme Programming e Scrum.

4.1. – Scrum

Scrum é um framework estrutural que está sendo usado para gerenciar o desenvolvimento de produtos complexos desde o início de 1990. Em torno de 55% dos desenvolvedores usam Scrum (VERSIONONE, 2013). Scrum não é um processo ou uma técnica para construir produtos; em vez disso, é um framework dentro do qual você pode empregar vários processos ou técnicas. O Scrum deixa claro a eficácia relativa das práticas de gerenciamento e desenvolvimento de produtos, de modo que você possa melhorá-las. Ou seja, ele simplesmente fornece uma estrutura para entrega, mas não diz como fazer práticas específicas, deixando isso para a equipe determinar (Figura 6). (RIBEIRO, 2015).

Figura 6 – Metodologia Scrum,



Fonte: Ribeiro (2015)

O framework Scrum consiste nos times do Scrum associadas a papéis, eventos, artefatos e regras. Cada componente dentro do framework serve a um propósito específico e é essencial para o uso e sucesso do Scrum. As regras do Scrum integram os eventos, papéis e artefatos, administrando as relações e interações entre eles.

4.2. – X.P. (Extreme Programming)

O primeiro projeto Extreme Programming (XP) foi iniciado 6 de março de 1996. XP é uma metodologia para desenvolvimento de sistema ágil, com qualidade e que atenda às necessidades do cliente. Alguns praticantes conceituam XP como a prática e a perseguição da mais clara simplicidade, aplicado ao desenvolvimento de software. Uma metodologia voltada para projetos cujos requisitos mudem com frequência, utilizem desenvolvimento orientado a objetos, equipes de até 12 desenvolvedores e desenvolvimento incremental. A XP Busca o máximo de valor a cada dia de trabalho da equipe para o seu cliente. Em um curto espaço de tempo o cliente terá um produto que possa ser utilizado, podendo aprender com o mesmo e reavaliar se o que foi desenvolvido é realmente o desejado. (TELLES, 2004 apud OLIVEIRA, 2009).

De acordo com Sbrocco (2012) essa metodologia, assim como outras abordadas neste estudo, tem como meta atender as necessidades dos clientes com qualidade e da forma mais simples possível. Por ser uma metodologia ágil, busca desenvolver os projetos rapidamente e entregá-los dentro de um prazo predefinido, mesmo que os requisitos mudem com frequência. A XP dá preferência ao desenvolvimento orientado a objetos e permite trabalhar com pequenas equipes de até 12 desenvolvedores (programadores), além de outras funções. Utiliza um modelo incremental, ou seja, na medida em que o software é utilizado, novas melhorias são implementadas. Desta forma, o cliente sempre tem um produto a ser utilizado e testado, possibilitando ao desenvolvedor conhecer e aprender com os processos da empresa ou projeto a ser construído, bem como o cliente avaliar se o que foi proposto foi desenvolvido exatamente.

Trata-se de uma metodologia flexível que pode ser adaptada a projetos distintos. Porém, caso se observe a necessidade de um maior formalismo e retrabalho, o uso dessa metodologia deve ser reavaliado.

Na metodologia XP pode ocorrer alterações em suas concepções e, logo, é habitual se deparar com modificações. A adaptação ao ambiente de desenvolvimento deve ser levada em conta, se um valor trouxer mais prejuízos do que benefícios é necessário reavaliar a utilização desta metodologia. A XP é organizada em torno de um conjunto de valores e práticas que atuam perfeitamente para assegurar um alto retorno do investimento efetuado pelo cliente.

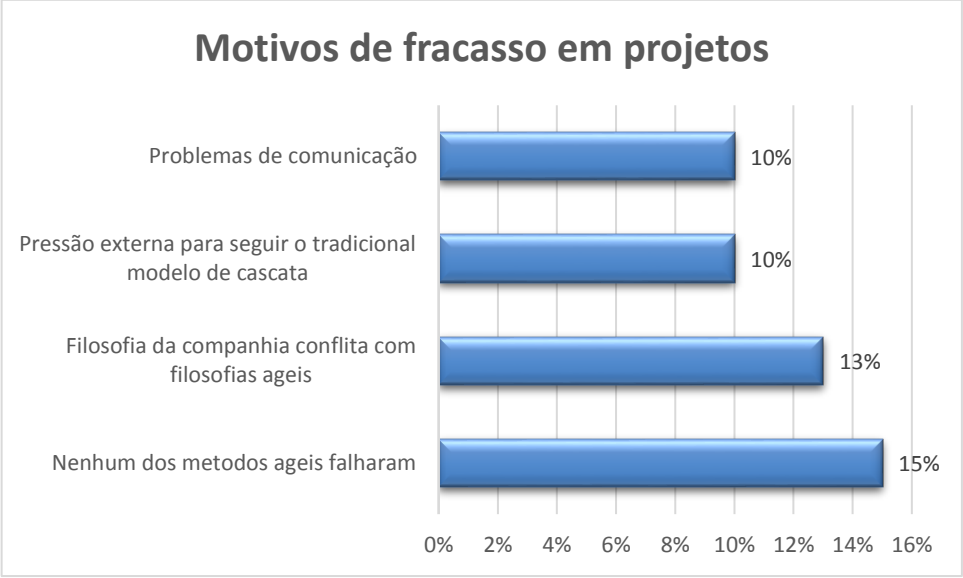
5. Desafios de construção de Software com Qualidade utilizando de metodologias ágeis

De acordo com Kruchten (2003), diferentes projetos de desenvolvimento de software falham de formas diferentes - e, infelizmente, muitos deles falham - mas é possível identificar vários sintomas comuns que caracterizam esses tipos de projetos:

- Incompreensão das necessidades do usuário final;
- Inabilidade para lidar com requisitos variáveis;
- Módulos que não se ajustam;
- Software difícil de manter ou estender;
- Descoberta tardia de sérias imperfeições do projeto;
- Baixa qualidade de software;
- Desempenho inaceitável de software.

Ao mesmo tempo, segundo pesquisa realizada pela VersioOne (2013), empresa de software que vende ferramenta de gerenciamento de métodos ágeis, em que foram questionados 3100 desenvolvedores pelo mundo, onde foi atestado que 23% dos fracassos eram devido a incapacidade da empresa que produz o software (Pressão externa + Filosofia da companhia) (Figura 7). Ainda segundo a pesquisa, 53% dos desenvolvedores diz que, entre as maiores dificuldades de adoção de metodologias ágeis, a dificuldade de empresa em mudar sua cultura organizacional.

Figura 7 – Motivos dos fracassos em desenvolvimento usando metodologias ageis



Fonte: http://stateofagile.versionone.com/ (acessado em 05/10/2016)

Como agravante, os problemas de software são de 100 a 1.000 vezes mais custosos de encontrar e reparar após a distribuição do que antecipadamente (Kruchten, 2003). Por esta razão, é importante avaliar continuamente a qualidade de um sistema com respeito à sua funcionalidade, confiabilidade, desempenho da aplicação e desempenho do sistema. Verificar a funcionalidade de um sistema - o volume de atividade de teste - envolve criar testes para cada cenário chave, cada qual representando algum aspecto do comportamento desejado do sistema. Verificar a qualidade de software oferece várias soluções para as causas de origem de problemas no desenvolvimento de software:

- Expõe inconsistências de requisitos, construções e implementações;
- Consciência de requisitos não funcionais e suas dificuldades de implementá-los (como a dificuldade de implementar método ágil em uma equipe não ágil);
- Testes e verificação são concentrados nas áreas de maior risco, aumentando assim a qualidade e efetividade dessas áreas;
- Deficiências são identificadas cedo, reduzindo radicalmente o custo para fixá-los;
- Ferramentas de teste automatizadas fornecem teste para funcionalidade, confiabilidade e desempenho.

Os desafios de se desenvolver softwares vão muito mais além do que problemas de processos e procedimentos. Trabalhar com expectativas, transferir e compartilhar conhecimento, motivação e um bom ambiente são exemplos de aspectos que devem ser considerados muito importantes no desenvolvimento de um software. Cada vez mais fica claro que o foco de pontos a melhorar e a melhoria contínua provem e depende das pessoas comprometidas com o desenvolvimento do software. Isto nos eleva a um novo patamar na cultura de desenvolvimento de software, onde, tanto quanto a Ciência de Software é considerada uma área Exata, sua aplicabilidade se demonstra cada vez mais uma área Humana.

6. Considerações Finais

Conclui-se que a qualidade é consequência dos processos (metodologias e consciência de seu uso), das pessoas e da tecnologia, e que a aplicação de boas práticas de qualidade de software é essencial para a melhoria no processo de desenvolvimento de sistemas. Sem a aplicação adequada de metodologias, os produtos de software continuarão imperfeitos e os clientes insatisfeitos. Observa-se que existem diversas formas de se construir sistemas de software, o qual cada empresa busca adequar a metodologia a sua realidade e necessidade e, também, a realidade e necessidade do cliente. Cabe ainda a cada empresa observar e atentar-se ao mercado, suas características e principalmente seus requisitos não funcionais: afinal, é não faz sentido implementar uma metodologia ágil em sua equipe se a mesma não tem princípios ágeis.

A qualidade dos softwares está condicionada aos recursos e processos que o produzem. Os processos devem ser identificados com clareza, devem ser documentados e repetidos para toda nova produção. Considera-se que uma empresa de informática (ou departamento de informática em qualquer instituição) poderá ter produtos de qualidade se, na sua administração, os recursos e processos forem identificados e utilizados seguindo uma metodologia de acordo com sua necessidade. Cada empresa, portanto, definirá seu método para a construção de softwares, baseando-se no seu quadro de pessoal técnico, sua experiência, competência e habilidades, e a plataforma e ferramentas disponíveis para auxílio aos processos.

7. Referências

SBROCCO, José Henrique Teixeira de Carvalho Sbrocco. Metodologias ágeis: engenharia de software sob medida, Paulo Cesar de Macedo. -- 1. ed. -- São Paulo: Érica, 2012.

CROSBY, Philip Bayard. Quality is Free. First Edition. New York: Editor McGraw-Hill.

FERREIRA, Aurélio Buarque de Holanda. Mini Aurélio Dicionário da Língua Portuguesa. 8ª Edição. São Paulo: Editora Positivo.

GARVIN, David A. Managing quality. First Edition. New York: Editor The Free Press.

PRESSMAN, Roger S. Engenharia de Software - Uma abordagem profissional. 7ª Edição. New York: Editora McGraw-Hill.

SOMMERVILLE, Ian. Software Engineering. 9ª Edit. New York: Editor Person Education.

POMPILHO, S. Análise Essencial Guia Prático de Análise de Sistemas. Rio de Janeiro: Editora Ciência Moderna, 1995.

TONSING, Sérgio Luiz. Engenharia de Software - Análise e Projeto de Sistemas. 2ª Edição Rio de Janeiro: Editora Ciência Moderna, 2008.

KRUCHTEN, Philippe. Introdução ao RUP - Rational Unified Process. Rio de Janeiro: Editora Ciência Moderna, 2003.

SOARES, Michel dos Santos. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. 2004. Disponível em: < http://www.dcc.ufla.br/infocomp/artigos/v3.2/art02.pdf> acesso em: 10 setembro 2016.

SCHWABER, Ken. Guia do Scrum – Guia definitivo para o Scrum. 2013. Disponível em: < https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide-Portuguese-BR.pdf> acesso em: 09 setembro 2016.

VERSIONONE. State of Agile Survey. 2013.

PAULA FILHO, Wilson de Pádua. **Engenharia de software:** fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: Editora LTD, 2005.

Gerenciamento de projetos com métodos ágeis / Rafael Dias Ribeiro, Horácio da Cunha e Sousa Ribeiro. Rio de Janeiro: [s.n.], 2015. Disponível em: < http://189.16.45.2/CONGRESSO/PEDAGOGIA/congresso2013/19.pdf>.

César Biancolino, Marco Terlizzi. Projeto de Software no Setor Bancário: Scrum ou Modelo V , 2014. Disponível em: < http://www.anpad.org.br/periodicos/arq_pdf/a_1495.pdf> acesso em: 02 outubro 2016.

Eduardo Oliveira; Cayley Guimarães. Motivação para utilização da Extreme Programming. Disponível em: <http://www.anpad.org.br/periodicos/arq_pdf/a_1495.pdf> acesso em: 03 outubro 2016.

Giovane Roslindo Kuhn; Vitor Fernando Pamplona. Apresentando XP. Encante seus clientes com Extreme Programming. Disponível em: http://vitorpamplona.com/deps/papers/2004_ApresentandoXP.pdf> acesso em: 03 outubro 2016.

Extreme Programming: A gentle introduction Disponível em: http://www.extremeprogramming.org/pdf> acesso em: 07 outubro 2016.