METODOLOGIAS DE DESENVOLVIMENTO DE *SOFTWARE*: UMA ANÁLISE NO DESENVOLVIMENTO DE SISTEMAS NA WEB

Fernando Gonçalves de Oliveira, João Manuel Pimentel Seabra

Resumo: No artigo apresentado foi comentado sobre temas relacionados à engenharia de *software* e aplicação de metodologias de desenvolvimento de *software*. Todos esses aspectos foram desenvolvidos ao longo do texto como pressuposto de obter conhecimento sobre o desenvolvimento de *software* e sua melhoria ao longo do tempo, além de apresentar uma aplicação da engenharia de *software* no desenvolvimento *web*. O assunto discutido neste artigo científico é de extrema importância para os profissionais que fabricam e as empresas que recebem esses *softwares*, diante disso, a pesquisa científica foi desenvolvida baseada em diversas literaturas que discorrem sobre o tema apresentado, tendo como finalidade esclarecer como acontece o desenvolvimento de um *software* e a relevância da metodologia para a construção de *software*.

Palavras-Chave: Engenharia; Metodologias; *Software*.

Abstract: *This work will be presented issues related to software engineering methodologies and application in software development. All these aspects have been developed throughout the text as a condition of obtaining knowledge about software development and its improvement over time, and present an application of software engineering at web development. It is thought that the subject discussed in this research paper is extremely important for professionals and companies that manufacture these receiving software. Scientific research has been developed based on several literatures that talk about the issue presented, whose purpose is to clarify how the development of a software and relevance of the methodology for software construction.*

Keywords*: Engineering; Methodologies; Software.*

Introdução

Este estudo tem como tema a explanação para adoção de uma metodologia de desenvolvimento de *software*, abordando o tema por meio de pensamentos de autores que delineiam as bases para estabelecimento dos métodos utilizados para produção de um produto de *software*.

Os *softwares* são desenvolvidos por equipes multidisciplinares, este processo produtivo deve ser dividido em etapas e atividades, indo desde o levantamento inicial até a aceitação do produto pelo usuário. A engenharia de *software* cria métodos, técnicas e ferramentas para tornar este processo mais eficiente e econômico (SEABRA, 2013).

Assim, a pesquisa também apresenta um comparativo entre a engenharia de *software* e o desenvolvimento de sistemas para a *web*, ponto importante a ser questionado, no que essas duas áreas do desenvolvimento de sistemas se complementam.

A engenharia de *software* se encontra estabelecida em diversas áreas. Sejam elas mais tradicionais ou recentes, cujo domínio é pouco conhecido ou altamente mutável, diante desse

exposto, o presente artigo fez uma revisão de algumas metodologias de desenvolvimento de *software*, destacando seus pontos fortes e fracos, assim como os pontos em comum entre elas e o desenvolvimento de sistemas para a *web*.

Segundo Pressman e Lowe (2009), a web se tornou uma indispensável tecnologia em diversas áreas, porém falar em dependência seria algo "pequeno", visto que a execução de nossas atividades cotidianas foram modificadas por novas tecnologias, como o *e-learning*, *e-commerce*, portais de informação, *blogs* e *internet bankings*.

A implantação de sistemas web advém da necessidade da rápida disseminação de informação e da esparsa distribuição geográfica de seus usuários, sendo utilizada a Internet como ferramenta de suporte. Assim, foi gerada uma nova dependência das organizações a esta nova ferramenta de trabalho o que causou uma grande revolução nos padrões de relacionamento entre os envolvidos no processamento da informação.

> Entretanto, esta evolução também gerou novas preocupações. Os clientes podem estar em locais diversos e desejam ter comodidade de acesso a qualquer horário, o que exige que os sistemas estejam disponíveis 24 horas. A grande facilidade de acesso exige que eles atendam a grandes volumes de usuários simultaneamente e tenham mecanismos sofisticados de segurança, pois podem ser alvos de ataques maliciosos (LEMOS, 2009).

Esta pesquisa objetiva descrever as metodologias em uso no desenvolvimento de sistemas em regime de fábrica de *software* e a sua eficácia nos seus cenários de implantação a fim de melhorar os seus produtos, prazos e custos.

Nesse sentido, o pressuposto para escolha do tema foi a relevância da engenharia de *software* e sua metodologia no desenvolvimento de sistemas para a *web*. Abordar esse assunto é uma maneira de entender seu funcionamento e destacar seus principais aspectos como o estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de *software*.

Aspectos do desenvolvimento de *software*

O desenvolvimento de *software* objetiva a criação de sistemas de *software* que correspondam às necessidades de clientes e usuários (VASCONCELOS, 2006). Desta forma, se torna fundamental que se realize uma correta especificação dos requisitos do *software* para se obter o sucesso do processo. Dessa forma, é cada vez mais utilizado dentro das organizações o papel do analista de requisitos, desempenhando um papel crucial importância.

Com isso surgem, as novas Metodologias de Desenvolvimento de *Software* (MDS), que dividem o processo de desenvolvimento de *software*, a fim de organizá-lo e facilitar seu entendimento. Assim, segundo Souza Neto (2004) e Soares (2004), dividem-se em duas áreas de atuação:

a) desenvolvimento "tradicional", a qual se fundamenta na análise e no projeto, que conserva tudo em documentação, no entanto, não é vantajoso para mudanças; e

b) desenvolvimento ágil, baseado em código, inteiramente adaptável a mudanças nos requisitos, mas deficiente na esfera contratual e de documentação.

Engenharia de *software*

Maia (2007) define a engenharia de *software* como um processo para a produção organizada que utiliza uma coleção de técnicas predefinidas e convenções de notação. Para Sommerville (2008) a engenharia de *software* é uma área da engenharia que se ocupa de todos os aspectos produtivos do *software*, desde os estágios iniciais de especificação e entendimento do sistema até sua a manutenção depois que ele entrou em operação.

Seabra (2013a) esclarece a importância dos artefatos produzidos durante o processo de Engenharia de *Software* indicando que são saídas e entradas das atividades do desenvolvimento do produto, independente da abordagem utilizada. Reforça ainda que a utilização correta dos artefatos no processo leva a um melhor entendimento do produto final e a sua ausência ou má qualidade afeta diretamente nas fases posteriores da produção do *software*. Os artefatos produzidos no processo de desenvolvimento de *software* e seus relacionamentos são indicados no diagrama abaixo:

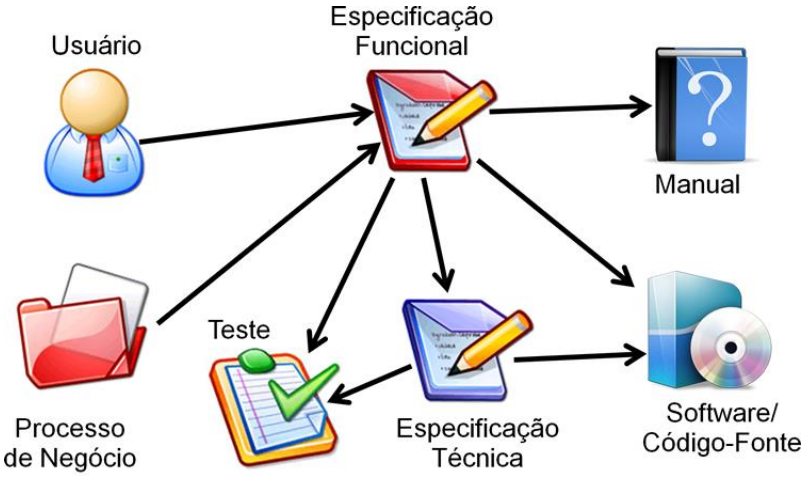

Figura 1 – Artefatos produzidos no processo de desenvolvimento e seus relacionamentos.

Fonte: (SEABRA, 2013a).

Podemos verificar que a engenharia de *software* se preocupa na implantação de métodos, ferramentas e técnicas no processo produtivo de sistemas, buscando a eficácia e eficiência dos recursos, melhorando a qualidade do produto final, bem como reduzindo o prazo e custo de produção.

Fases do desenvolvimento de *software*

A engenharia de *software* provê técnicas para construção de um *software*. Os métodos compreendem uma matriz ampla que incluem comunicação, análise de requisitos, modelagem de projeto, construção, teste e suporte. Estes métodos dependem de uma série de princípios básicos que governam cada área de tecnologia e incluem atividades de modelagem e outras técnicas descritivas (PRESSMAN, 2015).

Segundo SOMMERVILLE (2008), não há processo ideal de desenvolvimento, e muitas organizações desenvolvem seu próprio processo. Porém os processos são desenvolvidos para explorar as capacidades das pessoas da organização e as características específicas dos sistemas que estão em desenvolvimento. Por exemplo, para um sistema crítico um processo bem definido é requerido, porém para sistemas com mudanças rápidas de requisitos um processo mais efetivo deveria ser ágil e flexível. No entanto, algumas atividades em comum são encontradas em todos os processos:

a) Especificação de *software*: A funcionalidade do *software* e suas constantes de operação são definidas;

b) Projeto de *software* e implementação: O *software* será produzido de acordo com seu projeto;

c) Validação de *software*: O *software* deve ser validado para garantir que executa a necessidade do cliente;

d) Evolução do *software*: O *software* deve evoluir para atender às necessidades de mudança do cliente.

Metodologias de desenvolvimento de *software*

As metodologias de desenvolvimento de *software* são criadas por meio da utilização coordenada de métodos, ferramentas e procedimentos, com foco na produção de um produto de *software* e seus artefatos relacionados. Estas metodologias buscam o aprimoramento da utilização dos recursos, buscando reduzir custos e tempo de execução e a melhoria da qualidade do produto final.

Como uma metodologia é um conjunto de métodos, tem-se a definição de método e qual o seu objetivo. Ou seja,

> É (são) o(s) procedimento(s) necessário(s) a ser (em) adotado(s) para atingir um objetivo. Já uma técnica, pode ser compreendida como sendo um modo apropriado de se investigar sistematicamente um universo de interesse ou domínio do problema. Para tanto, utiliza-se de uma notação. Como exemplo de técnica, temos: Análise estruturada, Análise Essencial, Projeto Estruturado, Análise Orientada a Objetos (SOARES, 2004).

A metodologia utilizada no desenvolvimento de um determinado projeto deve ser escolhida de acordo com as características do ambiente e do produto a ser desenvolvido, assim como dos métodos e ferramentas que serão utilizadas e dos controles e produtos intermediários pretendidos.

Modelos Prescritivos

Pressman (2015) define os modelos prescritivos como aquele que luta por estrutura e ordem no desenvolvimento de *software*. As atividades e tarefas ocorrem sequencialmente com diretrizes definidas para o progresso. São chamados assim, pois são prescritos como um conjunto de elementos de processo, cada um com o seu fluxo de trabalho específico, onde os elementos de processo interagem.

Os modelos prescritivos também são chamados de modelos clássicos e refletem uma preocupação em definir a fronteira do sistema antes de encaminhar para o desenvolvimento, sendo por diversas vezes modelos que não têm aderência às mudanças, exigindo uma grande quantidade de retrabalho quando a necessidade do cliente é alterada. Estes modelos exigem uma grande quantidade de profissionais envolvidos, pois são produzidos muitos artefatos durante a execução.

Metodologia em cascata

A metodologia em cascata baseia-se na execução sequencial de atividades. Definido por Calçado (2007) como um modelo linear em que cada passo deve ser completado antes que o próximo passo possa ser iniciado. Sommerville (2008) indica ainda que esta metodologia o primeiro modelo de desenvolvimento de *software* publicado e deriva do processo geral de engenharia de sistemas de Royce.

Vasconcelos (2006) descreve as etapas de forma geral. A figura 2 ilustra os seus relacionamentos:

a) Definição dos requisitos: Deve-se identificar juntamente aos usuários do *software*, os serviços, metas e restrições impostas ao sistema.

b) Projeto do sistema e do *software*: Os requisitos identificados são "mapeados em componentes de hardware e *software*, de modo que o sistema possa ser posteriormente implementado". Ocorre, ainda, a fundamentação da arquitetura geral do sistema;

c) Implementação e testes unitários: Por meio da linguagem de programação, implementa-se o projeto de *software* em unidades de programas;

d) Integração e teste do sistema: A fim de se garantir o atendimento a todos os requisitos de *software*, as unidades de programas são integradas e testadas como um sistema completo. Os testes devem ser realizados à medida que os componentes individuais do sistema são integrados. Ao final do processo é recomendado um teste de toda a aplicação.

e) Operação e manutenção: O sistema é instalado e colocado em operação. Posteriormente, é evoluído quando surgirem falhas no sistema ou quando surgirem novas necessidades do cliente, voltando ao fluxo inicial do processo para início da manutenção.

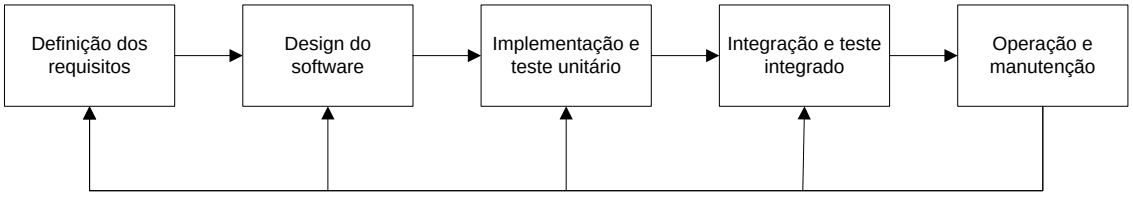

Figura 2 - Metodologia de Desenvolvimento em Cascata

Segundo Pressman (2015), os problemas da adoção deste modelo são:

1) Os projetos raramente seguem o fluxo proposto pelo modelo, este modelo pode acomodar iterações, porém realiza de forma não-natural. Como resultado, as mudanças causam confusão no procedimento da equipe de projeto;

2) É complicado para o cliente explicitar todos os requisitos de início. Este modelo tem dificuldade em acomodar as incertezas naturais que existem no início da maioria dos projetos;

3) O cliente deve ter paciência, visto que uma versão funcional do programa não estará pronta até o final do cronograma. Uma falha não detectada previamente pode ser desastrosa.

Nesse modelo, as mudanças são desencorajadas após a fase de definição de requisitos, pois como o escopo do produto final já foi delimitado, qualquer mudança detectada nas fases posteriores tem um forte impacto em retrabalho, aumentando sobremaneira os custos e os prazos do projeto.

Modelo Incremental

O modelo incremental é uma evolução da metodologia em cascata. Segundo Sommerville (2008), o cliente identifica as funcionalidades do *software*, priorizando os que serão entregues primeiro. Depois, um número de entregas é planejada, de forma que cada entrega provê uma parte do sistema com a funcionalidade solicitada. Uma vez que o incremento está entregue, este pode ser colocado em produção. À medida que as novas iterações vão sendo entregues, elas são adicionadas ao produto em produção, de forma que este sempre evolua em direção ao produto final planejado pelo cliente.

O refinamento das versões intermediárias do *software* por parte do cliente permite que este receba o seu valor agregado mais rapidamente e surjam necessidades mais maduras após a sua utilização, permitindo à equipe de desenvolvimento uma maior precisão na solução do problema.

Para Vasconcelos (2006), a funcionalidade do sistema é disponibilizada mais rapidamente, sendo os primeiros incrementos atuando como um modelo que auxilia a elaboração dos requisitos para incrementos finais. Assim, os riscos de falhas no projeto são amenizados e os serviços de maior criticidade do sistema tendem a receber maior quantidade de testes.

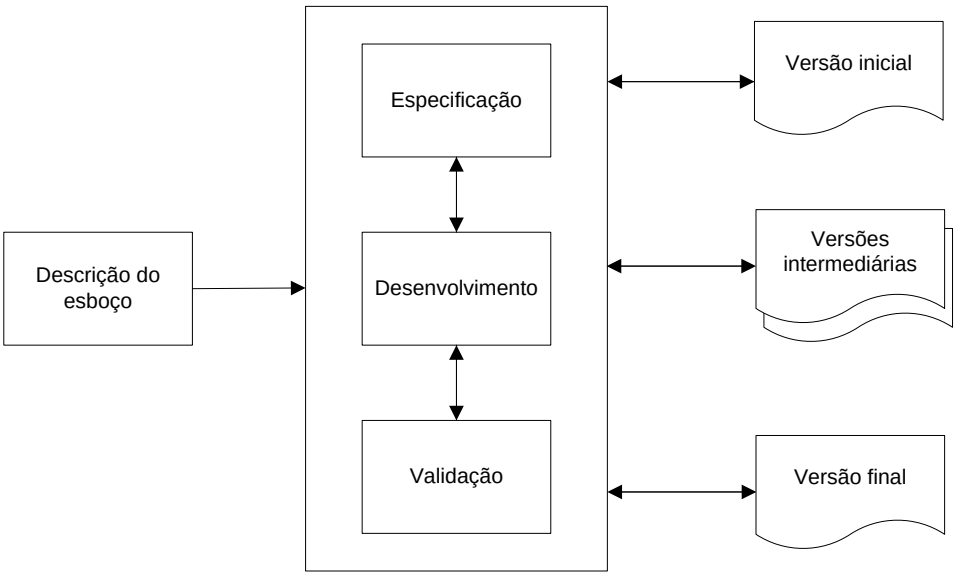

Figura 3 - Metodologia de Desenvolvimento Incremental

Rational Unified Process – RUP

O *Rational Unified Process* (RUP) é um processo, de propriedade da IBM disponível como um produto para venda e, origina-se do Processo Unificado (UP), objetivando fornecer as melhores práticas de desenvolvimento de *software* para as organizações (SEABRA, 2013b).

O produto é gerado no RUP por meio de evoluções sucessivas, permitindo o entendimento e solução do problema com linhas de base bem definidas, facilitando o entendimento do objetivo de cada iteração a ser realizada em sua execução, garantindo a produção de um produto de qualidade, atendendo às necessidades do usuário com prazos e custos previsíveis.

Segundo Sommervile (2008), o RUP possui 03 perspectivas de execução, sendo elas:

a) Dinâmica: Esta perspectiva mostra as fases do processo ao longo do tempo de execução do projeto;

b) Estática: Esta perspectiva mostra como as atividades do processo são executadas pelos envolvidos no projeto;

c) Prática: Esta perspectiva busca ensinar as melhores práticas para serem utilizadas durante a execução do processo.

Em sua perspectiva dinâmica, o RUP é dividido em 04 fases distintas, que objetivam entregar um determinado produto planejado para aquele momento. Caso o objetivo daquela fase ou iteração não tenha sido atingido, outra iteração é executada. Esta proposta dinâmica, provê aos usuários finais validarem versões parciais do produto, possibilitando um *feedback* mais eficiente e a identificação antecipada de problemas.

Na perspectiva estática, o RUP divide-se em 09 disciplinas que definem os fluxos de trabalho por áreas de conhecimento correlatas, podendo ser estas diretamente relacionadas ao trabalho de produção do *software* (Modelagem de Negócios, Requisitos, Análise e Design, Implementação, Teste e Implantação) e disciplinas que fundamentam e apoiam o trabalho do projeto (Gerenciamento de Configuração e Mudança, Gerenciamento de Projetos, Ambiente).

A perspectiva prática busca difundir entre os praticantes de RUP as melhores práticas de desenvolvimento de *software*. Segundo SEABRA (2013b), as melhores práticas implementadas buscam o desenvolvimento orientado ao negócio e são independentes e não sequenciais, sendo elas:

a) Adaptação do processo;

b) Equilíbrio as prioridades dos envolvidos;

c) Colaboração entre as equipes;

d) Demonstração de valor do produto iterativamente;

e) Nível de abstração elevado;

f) Foco contínuo na qualidade.

O relacionamento entre as disciplinas, fases e o esforço correspondente para atuação são descritas por meio do gráfico disponibilizado na figura 4.

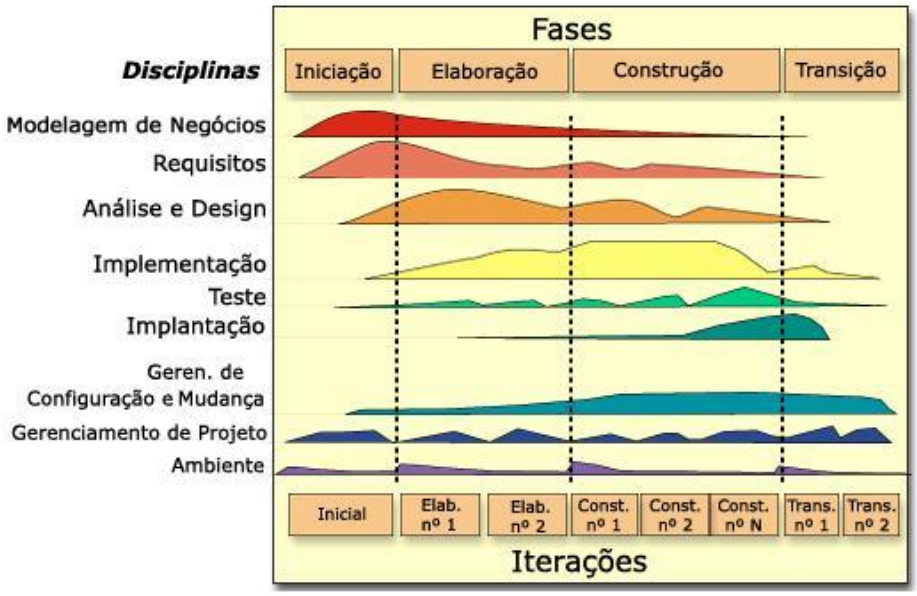

Figura 4 - Relacionamento entre fases, disciplinas e esforço.

Metodologias ágeis

Para Sabbagh a agilidade surgiu a partir do chamado Manifesto Ágil, criado a partir de uma reunião, em 2001, de 17 estudiosos de processos de desenvolvimento de *software*. Este manifesto prega mais que uma metodologia de *software* e sim, uma mudança de paradigmas e comportamento na indústria, buscando melhorar a relação entre os indivíduos, resposta a mudanças, o *software* em funcionamento e a colaboração com o cliente. O Manifesto Ágil tem a seguinte redação:

> Estamos descobrindo maneiras melhores de desenvolver *software* fazendo-o nós mesmos e ajudando outros a fazê-lo. Através deste trabalho, passamos a valorizar:
>
> - **Indivíduos e interação entre eles** mais que processos e ferramentas
>
> - ***Software* em funcionamento** mais que documentação abrangente
>
> - **Colaboração com o cliente** mais que negociação de contratos
>
> - **Responder a mudanças** mais que seguir um plano
>
> Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda. (SABBAGH, op. cit.)

O desenvolvimento ágil aparece como uma das soluções para a influência das mudanças ao longo do projeto de desenvolvimento de *software*. Para os estudiosos da abordagem ágil, o cliente adquire conhecimento sobre seu produto durante o seu desenvolvimento, gerando mudanças ao longo deste processo. Este processo busca dividir o projeto em pequenas entregas ao longo do cronograma, onde o cliente pode avaliar o desenvolvimento, agregar valor ao negócio, priorizar e solicitar mudanças de funcionalidades.

Para Pressman (2013), agilidade é mais que resposta às mudanças. Ela incorpora uma filosofia ditada pelo manifesto ágil com ênfase na rápida entrega de uma aplicação operacional, em detrimento dos produtos de trabalho intermediários, inclui o cliente como parte da equipe de desenvolvimento e reconhece que o plano de projeto deve ter flexibilidade para atender às mudanças que acontecem ao longo do desenvolvimento.

Scrum

Sabbagh (2013) define o Scrum como:

> [...] framework ágil, simples e leve, utilizado para a gestão do desenvolvimento de produtos complexos imersos em ambientes complexos. Scrum é baseado no empirismo e utiliza uma abordagem iterativa e incremental para entregar valor com frequência e, assim, reduzir os riscos do projeto. (SABBAGH, op. cit.)

Pressman (2015) indica que o *Scrum* enfatiza o uso de um conjunto de padrões de processo de *software* que provê efetividade para os projetos com cronogramas apertados, mudanças de requisitos e criticidade de negócio.

Sabbagh (2013) define os papéis do Scrum da seguinte forma:

> **1)** ***Time*** **de Desenvolvimento:** Representado por uma equipe multidisciplinar, responsável por realizar o desenvolvimento do produto;
>
> **2)** ***Product Owner:*** Responsável por garantir o retorno sobre o investimento do produto para os clientes, ele é quem define e gerencia o produto, tomando as decisões de negócio relativas às necessidades dos interessados no projeto.
>
> **3)** ***Scrum Master*****:** Facilitador do trabalho da equipe do *Scrum*, ele é o responsável por garantir a execução do trabalho de desenvolvimento, removendo impedimentos, facilitando a comunicação, promovendo mudanças organizacionais e assegurando o entendimento do *Scrum*.

Nessa metodologia, os projetos são divididos em funcionalidades entregáveis, que produzam valor para o cliente. Os entregáveis são priorizados pelo *Product Owner*, e colocados em um *Product Backlog*, que representa a sequência de desenvolvimento das funcionalidades do sistema. No início de cada ciclo, chamado de *Sprint*, que dura de 2 a 4 semanas, as funcionalidades são colocadas em um *Sprint Backlog*, que representa os produtos que serão entregues para o cliente ao final do *Sprint*. Para acompanhamento do projeto, diariamente são realizadas as *Scrum Meetings*, que são reuniões rápidas de *feedback* objetivando a priorização das atividades do novo dia de trabalho. A figura 5 representa o ciclo de uma *Sprint* do *Scrum*.

Figura 5 – Ciclo do *Scrum*

Para Sabbagh (2013), o *Scrum* está sendo adotado com sucesso por diversos tipos de organização, pois seu uso não se limita a desenvolvimento de *software*, podendo ser utilizado em projetos de todos os tamanhos e tipos, sendo listadas como suas principais vantagens de utilização:

1. Entregas frequentes de produtos, retornando o investimento dos clientes;
2. Minimização de riscos do projeto;
3. Aumento de qualidade do produto gerado;
4. As mudanças podem ser utilizadas como vantagem competitiva;
5. Visibilidade por parte do cliente do progresso do projeto;
6. Redução de desperdício de tempo e investimento;
7. Aumento de produtividade.

Extreme Programming – XP

O XP é definido por Teles (2006) como um processo que busca garantir que o cliente receba o máximo de retorno sobre o investimento a cada dia de trabalho executado pela equipe. Os valores e práticas que atuam no processo de forma coesa asseguram que o cliente receba o máximo de retorno diário sobre o seu investimento.

O XP é um processo altamente recomendado no conjunto das situações abaixo:

1) Desenvolvimento interativo;
2) Projetos com requisitos flutuantes ou vagos;
3) *Time* de desenvolvimento de até 12 programadores;
4) Sistemas desenvolvidos sob o paradigma de Orientação a Objetos;

A XP tem ênfase em um rápido desenvolvimento do projeto, assim visa garantir a satisfação do cliente, sendo favorecida para o cumprimento das estimativas acordadas. A condução de seus trabalhos respeita quatro valores: "coragem, comunicação, simplicidade e *feedback*" (BECK, 1999a).

Teles (2006), lista as práticas do XP:

1) Cliente presente: O cliente deve participar ativamente do processo de desenvolvimento, conduzindo o processo por meio de seus feedbacks;

2) Jogo do Planejamento: A cada iteração, o cliente avalia as funcionalidades que serão implementadas e a equipe estima sua velocidade de desenvolvimento com uma unidade de medida especial chamada de "ponto";

3) *Stand Up Meeting*: A equipe do projeto se reúne rapidamente uma vez por dia, normalmente no início ou no fim do dia, para avaliar o que foi executado e planejar o restante do trabalho;

4) **Programação em par:** Cada funcionalidade será desenvolvida por um par de desenvolvedores, utilizando a mesma máquina, permitindo que o código seja revisado permanentemente;

5) **Desenvolvimento Guiado pelos Testes:** Os desenvolvedores realizam a codificação somente após escrever testes para aquela funcionalidade, com isso tem um entendimento melhor da necessidade o que melhora sobremaneira a qualidade do código gerado;

6) ***Refactoring*:** O *software* é constantemente modificado internamente, sem alterar as funcionalidades implementadas, buscando otimizar a sua manutenção;

7) **Código Coletivo:** Os desenvolvedores podem realizar modificações em qualquer parte do código sem pedir permissão, visto que o código é coletivo;

8) **Código Padronizado:** São estabelecidos padrões de codificação para tornar o código mais homogêneo e possa ser entendido por toda a equipe;

9) **Design Simples:** O código é sempre o mais simples possível, buscando atender o mais rápido possível à necessidade imediata, sem se preocupar em suportar possíveis evoluções futuras;

10) **Metáfora:** As ideias do projeto são transmitidas entre a equipe por meio de metáforas, buscando simplificar a comunicação, tornando as ideias complexas mais simplificadas;

11) **Ritmo Sustentável:** Para manter a qualidade do produto, o XP recomenda que a equipe trabalhe oito horas por dia, evitando a utilização de horas-extras, buscando a plenitude da produtividade da equipe ao longo do dia;

12) **Integração Contínua:** Os códigos desenvolvidos são integrados diversas vezes ao dia para assegurar que o sistema como um todo esteja funcionando de forma harmoniosa.

13) **Releases Curtos:** A equipe produz um conjunto de funcionalidades em um pequeno espaço de tempo, permitindo que agregue valor mais rápido ao cliente;

Para Beck (1999b), o XP explora a redução do custo das mudanças a serem realizadas em um projeto, executando as atividades de planejamento, análise e design um pouco de cada vez durante todo o desenvolvimento do *software*. BECK (op cit.) explica também que os limites de sua aplicação ainda não são claros, recomendando sua aplicação primeiramente nos projetos terceirizados ou internos de tamanho pequeno a médio, onde os requisitos são vagos e modificados constantemente.

A engenharia de *software* e o desenvolvimento de sistemas para web

A construção de *software*s para web é uma atividade essencialmente multidisciplinar, pois envolve além dos profissionais já utilizados em outras áreas de desenvolvimento, há uma

participação muito grande de atores das áreas de iteração homem-máquina, *webdesigners*, engenheiros de otimização de busca (SEO), entre outras especialidades.

Diante deste quadro, o desenvolvimento tradicional muitas vezes não atende às peculiaridades de desenvolvimento web. Desta forma, são buscadas novas abordagens de desenvolvimento, buscando uma maior agilidade na entrega do produto e, também, a inclusão destes novos papéis e atividades no processo.

Conforme descrito por Murugesan *et al*. (2001), falta rigor, abordagem sistêmica, controle de qualidade e garantia aos métodos utilizados para desenvolvimento de aplicações web. Deste problema, surge a necessidade de uma nova abordagem para disciplinar e introduzir novos métodos e ferramentas para o desenvolvimento, validação e implantação de sistemas web.

Esta nova área pode ser chamada de Engenharia Web, definida por Pressman e Lowe (2009), como uma proposta de um ágil, porém disciplinado framework para produção de aplicações web. Este framework seria composto pelas seguintes atividades:

1) Comunicação: Envolve a interação e colaboração do cliente e interessados e engloba a coleta de requisitos e outras atividades relacionadas;

2) Planejamento: Estabelece um plano incremental para o trabalho. Ele descreve como as ações irão ocorrer, como serão conduzidas as tarefas técnicas, os riscos associados, os recursos que serão necessários, os produtos que serão produzidos e o cronograma de trabalho;

3) Modelagem: Engloba a criação dos modelos que ajudarão o desenvolvedor e o cliente a entender melhor os requisitos e o projeto da aplicação;

4) Construção: Combina a geração do código-fonte, junto com os testes para verificar erros na aplicação;

5) Implantação: Realiza a entrega do incremento da aplicação para os clientes que validarão e providenciarão um feedback deste processo.

Para Kumar e Sangwan (2011), são os atributos da aplicação que definem o processo a ser utilizado. Se imediatismo e contínua evolução são atributos primários, deve ser utilizado um processo ágil. Caso o produto seja desenvolvido durante um longo período, um processo incremental deve ser utilizado. Para pequenos projetos, uma cascata tradicional pode ser utilizada, porém para projetos grandes e complexos, o processo em espiral pode ser mais apropriado.

Pressman e Lowe (2009) listam as melhores práticas para o desenvolvimento de aplicações web:

1) O tempo deve ser utilizado para entender o objetivo do produto e sua necessidade de negócio;
2) A interação dos usuários com a aplicação deve ser descrita por meio de cenários;
3) Um plano de projeto deve ser desenvolvido, mesmo que breve;

4) Deve ser realizada modelagem de pontos mais obscuros do projeto;

5) Os modelos devem ser revistos quanto à consistência e qualidade;

6) A tecnologia deve permitir que os componentes utilizados para a construção devem ser reutilizados quantas vezes forem possíveis;

7) Utilizar *design patterns* desenvolvidos para reutilização de componentes;

8) Os testes devem ser criteriosos antes da liberação para implantação.

Considerações finais

Os estudos apresentados neste trabalho evidenciam a necessidade da utilização de um processo sólido para produção de qualquer tipo de aplicações Web. A implantação de processos ágeis nos trouxe novas abordagens no desenvolvimento de *software*. No entanto, nos trouxe também mais problemas em relação à aplicação da engenharia de *software* nos projetos.

Esta pesquisa demonstra que o processo utilizado não é responsável direto pelo sucesso do projeto e sim, sua adequação ao ambiente onde foi implantado, atendendo à demanda do cliente da melhor maneira possível, seja com a entrega de um produto apenas ao final do cronograma ou com entregas sucessivas ao longo do projeto.

Verifica-se que os clientes não têm interesse no processo de produção da aplicação, descartando ou minimizando a importância dos artefatos e valorizando apenas o *software*. Este motivo dá-se principalmente pelo fato de que a documentação e o processo de desenvolvimento não agregar valor ao negócio e sim, a aplicação que objetiva resolver seus problemas negociais. Esta visão tem causado problemas em relação à aderência do produto às especificações e necessidades do cliente, bem como custo com retrabalho, atrasos e insucessos no projeto.

Como forma de ampliar os conhecimentos acerca do assunto sugere-se que se realize um levantamento dos processos utilizados para o desenvolvimento de *software* nas áreas pública e privada, relacionando com o tamanho dos projetos e sua taxa de sucesso. Esta pesquisa pode levantar um quadro atual do desenvolvimento no mercado brasileiro e sugerir novas soluções para atender às novas necessidades.

Referências

BECK, Kent. **Programação Extrema Explicada**. Bookman, 1999(a).

_____. **Embracing Change with Extreme Programming.** Computer. IEEE, vol 32, issue 10, p. 70-77, 1999(b).

CALÇADO, Vera Lúcia Xavier de Sales. **Influência da Utilização de Processo Unificado**, Testes e Métricas na Qualidade de Produtos de *Software*. [Dissertação]. Universidade de Brasília. Brasília: UnB, 2007.

GINIGE, Athula; MURUGESAN, San. "Web Engineering: An Introduction". IEEE Multimedia. Janeiro-Março 2001.

KUMAR, Sandeep; SATYAVEER, Sangwan. **Adapting the *Software* Engineering Process to Web Engineering Process.** International Journal of Computing and Business Research, vol 2, issue 1, 2011.

LEMOS, Roberto Duarte Pontual de. **Elementos Básicos para a Construção de Modelo de Gestão da Informação para Engenharia de *Software***: estudo de caso para atividade de requisitos no Serpro. [Dissertação]. Universidade de Brasília. Brasília 2009.

MAIA, Leticia Toledo. **Um Estudo sobre Aplicação de Técnicas de Inteligência Artificial e Engenharia de *Software* à Construção de um Sistema de Supervisão e Controle**. [Dissertação]. Universidade de Brasília. Brasília, 2007.

MURUGESAN, San et ali. **Web Engineering: a New Discipline for Development of Web-Based Systems.** In: _____; DESHPANDE, Yogesh. (Ed.) **Web Engineering - Managing Diversity and Complexity of Web Application Development.** Springer Berlin Heidelberg, 2001.

PRESSMAN, Roger S. ***Software* Engineering: A practitioner's approach.** 8ª Ed., McGraw-Hill, 2015.

_____; LOWE, David. **Web engineering : a practitioner's approach.** McGraw-Hill, 2009.

SABBAGH, Rafael. **Scrum: Gestão ágil para projetos de sucesso.** São Paulo : Casa do Código, 2013.

SEABRA, João. UML: Uma ferramenta para o design de *software*. Rio de Janeiro : Ciência Moderna, 2013a.

_____. **Aplicando as Melhores Práticas do PMBok no Planejamento para a Adaptação e Implantação do Rational Unified Process.** [Monografia de Pós-Graduação]. Unicesp. Brasília, 2013b.

SOARES, Michel dos S. **Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de *Software*.** Infocomp – Revista de Ciência da Computação. Lavras, v. 2, p. 8-13, 2004.

SOMMERVILLE, Ian. ***Software* Engineering.** 8ª Ed. Harlow, UK: Pearson Education, 2008.

SOUZA NETO, Oscar Nogueira de. **Análise Comparativa das Metodologias de Desenvolvimento de *Software*s Tradicionais e Ágeis**. [Monografia]. Belém-PA: Universidade da Amazônia, 2004.

TELES, Vinícius Manhães. **Extreme Programming**: Aprenda a Encantar seus Usuários Desenvolvendo *Software* com Agilidade e Alta Qualidade. São Paulo: Novatec, 2006.

VASCONCELOS, Alexandre Marcos Lins de; ROUILLER, Ana Cristina; MACHADO, Cristina Ângela Filipak; MEDEIROS, Teresa Maria Maciel de. **Introdução à Engenharia de *Software* e à Qualidade de *Software***. Universidade Federal de Lavras – UFLA. Lavras: UFLA/FAEPE, 2006.