METODOLOGIAS DE DESENVOLVIMENTO DE SOFTWARE

Ellen Lúcia de Moura[1]
Marcio Aurélio Ribeiro Moreira[2]

RESUMO

Este artigo apresenta um estudo bibliográfico sobre três metodologias de desenvolvimento de software: Cascata, RUP e XP. Todas têm o mesmo objetivo: organizar, padronizar códigos, diminuir riscos, documentar, auxiliar na administração, adequar-se a padrões de qualidade perante o desenvolvimento, porém o fazem de forma diferente. O artigo tem como foco mostrar as disciplinas que cada metodologia utiliza no ciclo de vida do desenvolvimento, apresentar uma análise comparativa e de viabilidade entre elas. O trabalho permite a conclusão que, a metodologia a ser usada depende principalmente do tamanho e da necessidade do projeto. O RUP, devido a sua ampla customização pode ser adaptado para projetos de qualquer escala e com distribuição geográfica. A XP é recomendada em equipes pequenas ou médias e locais pelo fato das reuniões serem feitas presencialmente.

Palavras-chave: RUP; XP; metodologias; engenharia de software; métodos ágeis.

ABSTRACT

This article presents a bibliographical study about three software development methodologies: Waterfall, RUP and XP. All have the same goal: to organize, standardize codes, mitigate risks, document, assist in the administration, conform to quality standards to the development, but do it differently. The article is focused on used disciplines by each method during the development lifecycle, presenting a comparative analysis and feasibility between them. The study allows the conclusion that the methodology to be used depends mainly on the size and the need for the project. The RUP, due to its wide customization can be tailored to projects of any scale with geographical distribution. The XP is recommended for small or medium sized teams and locals projects because of the face meetings.

Keywords: RUP; XP; methodologies; software engineering; agile methods.

[1] Aluna do curso de Especialização em Desenvolvimento de Aplicativos Web. Graduada no curso de Sistemas de Informação. Atua profissionalmente como Programador de Computador. E-mail: ellenlucia18@gmail.com.
[2] Professor Orientador, Bacharel em Ciência da Computação pela UFU. Especialista em Segurança da Informação pela Uniminas/Pitágoras. Atua como executivo, consultor em TI e especialista em projetos em várias empresas no mercado. E-mail: marcio.moreira@pitagoras.com.br.

1. Introdução

As empresas de desenvolvimento de sistemas frequentemente buscam se atualizar e se aperfeiçoar para conseguir atender a um mercado de constantes mudanças e exigências. Frente a isso, as corporações buscam as melhores ferramentas de trabalho para conseguir desenvolver produtos de qualidade e se destacarem das demais organizações.

No caso de desenvolvimento de software o que melhor garante seu sucesso é a forma como ele é feito, ou seja, a metodologia usada no seu desenvolvimento. A metodologia serve como um apoio, um "manual de como fazer" para o desenvolvedor de sistemas, visando entre outras coisas, a elaboração de requisitos, maior produtividade e redução de riscos. Atualmente existem várias metodologias de desenvolvimento no mercado.

Neste artigo foram escolhidas três metodologias de desenvolvimento de software para estudo bibliográfico: Cascata, RUP e XP.

Como produto deste espera-se mostrar as principais características das metodologias, uma comparação de seus ciclos de vida, suas atividades e uma análise de qual delas é a mais viável a ser seguida nos dias de hoje.

2. Metodologias de Desenvolvimento de Software

Neste capítulo será apresentado o fundamento teórico do estudo, mostrando as atividades realizadas nos ciclos de vida do desenvolvimento de software das metodologias: Cascata, RUP e XP.

2.1 Cascata (*waterfall*)

O primeiro processo de desenvolvimento de software foi o modelo cascata, chamado também de ciclo de vida clássico. Ele tornou-se conhecido na década de 70, por um artigo publicado por W. W. Royce e é referência para muitos outros modelos. Introduziu importantes qualidades ao desenvolvimento. Nele as atividades do processo de desenvolvimento são estruturadas numa cascata de forma linear e sequencial de forma que uma tarefa só poderá ter início quando a anterior tiver terminado, o desenvolvimento é conduzido de forma disciplinada como mostrado na Figura 1. (SOMMERVILLE, 2011)

O fato de ser dividido em etapas traz organização e orienta melhor as equipes de desenvolvimento antes de iniciar a codificação dos programas. Este modelo preza também o planejamento, principalmente no início, reduzindo assim a necessidade de planejamento contínuo conforme o andamento do projeto. E ao final de cada etapa, a equipe de projeto finaliza com uma revisão. (CARLOS, A. et al, 2010)

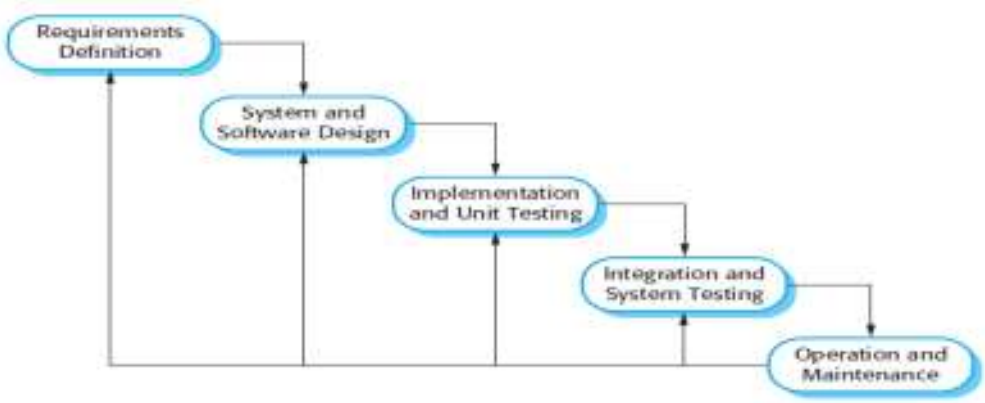

Figura 1 - Modelo Cascata
Fonte: Adaptada de SOMMERVILLE (2011)

2.1.1 Fases do modelo cascata

O modelo cascata é composto pelas seguintes fases:

a) **Levantamento de requisitos:** Definir objetivos, regras e restrições, indicar as funcionalidades e necessidades que o sistema terá. Gerar o documento de especificação do sistema que serve de base para o orçamento, cronograma entre outros.
b) **Análise:** Compreensão dos requisitos e das funcionalidades do software. Produz especificação documentada da descrição completa do comportamento do software.
c) **Projeto:** Descrição do contexto do software, indicar como o software deve ser implementado. Produz documentos de especificação técnica de cada parte do software, considerando questões como: interface com o usuário, armazenamento de dados.
d) **Codificação e Testes:** As unidades do software devem ser codificadas em qualquer linguagem de programação. Após a codificação as funcionalidades são testadas e corrigidas individualmente e em conjunto. Visa garantir se o código traz o resultado esperado.
e) **Entrega e manutenção:** O sistema é liberado para utilização e são oferecidos treinamentos aos usuários envolvidos. A manutenção se encarrega de corrigir os erros que serão encontrados, bem como ajustes e novas exigências conforme a evolução do sistema.

A metodologia em cascata é baseada em documentos. Isso abrange mais do que arquivos em texto, abrange também representações gráficas ou mesmo simulação.

Nesta metodologia a integração e os testes são feitos no fim do ciclo de desenvolvimento.

2.1.2 Desvantagens do modelo cascata

Somente é permitida a participação do cliente no início e no final do projeto, e é muito difícil o cliente passar todas as necessidades do sistema logo no início, outro problema é que ele só terá acesso ao sistema no final do projeto e só aí os erros serão encontrados. (PRESSMAN, 2010)

Nem sempre os prazos de entrega são cumpridos, pois segundo Pressman (2010) "Projetos reais raramente seguem o fluxo sequencial que o modelo propõe". A estrutura sequencial não permite voltar atrás para corrigir falhas.

2.2. RUP *Rational Unified Process* (ou Processo Unificado Racional)

O RUP é uma metodologia, um processo de engenharia de software que trabalha através de disciplinas e melhores práticas comprovadas, aumentando a produtividade da equipe de desenvolvimento de software.

É um processo proprietário criado pela Rational Software Corporation, adquirida pela IBM. Lembra o modelo em cascata, porém cada fase é composta de uma ou mais iterações, que em geral duram de uma a duas ou três semanas para serem feitas. Isto reduz o impacto de mudanças, pois o período é muito curto com menor risco de haver mudança na atividade em questão. É um processo aplicável a pequenos, médios e grandes projetos. Devido a sua ampla customização é possível que seja adaptado para projetos de qualquer escala. (VASCO, VITHOFT, ESTANTE, 2011)

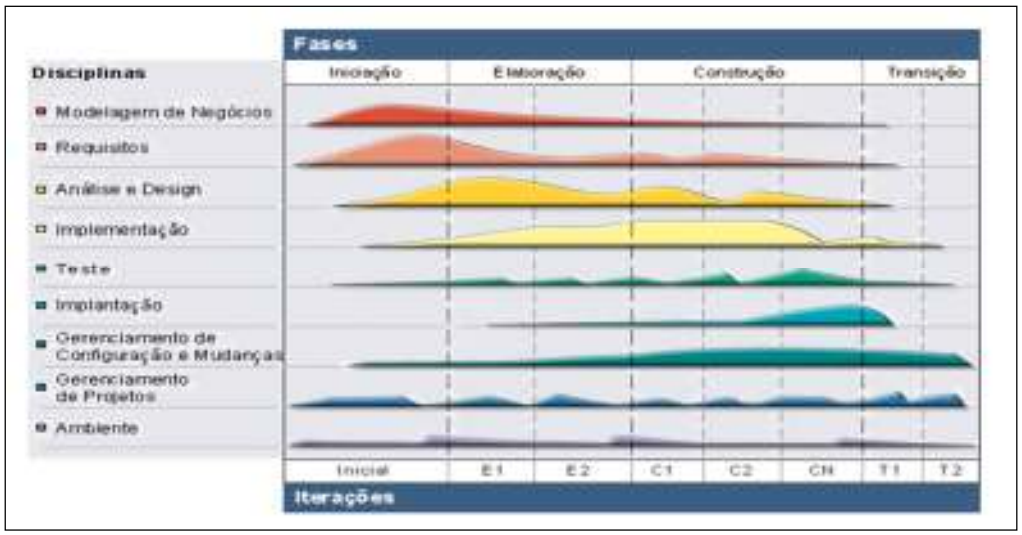

Figura 2 - Modelo básico de RUP - O gráfico mostra a ênfase que é dada em cada fase do RUP.
Fonte: VIGGIANO (2011)

O RUP é composto de quatro fases, como mostra a Figura 2, cada uma relacionada a um progresso do projeto. Na iniciação se define o escopo do projeto a partir de casos de uso de negócio, após isso se inicia a elaboração, com a definição das funcionalidades através dos casos de uso de sistema e a arquitetura a serem utilizadas. A partir daí é que o sistema começa a ser desenvolvido (fase de construção) e depois de pronto é implantado, os usuários são treinados e estes avaliam o produto (fase de transição). (MOREIRA, 2011)

Em conjunto com as fases, vêm as disciplinas que são as áreas de conhecimento utilizadas no RUP, cujas fases citadas anteriormente dão mais ou menos ênfase dependendo da importância e do momento dentro do projeto.

A Modelagem de Negócios tem o objetivo de avaliar a estrutura da organização como um todo para entender todos os problemas e derivar os requisitos necessários tendo assim a visão do negócio e o entendimento de como o software irá se ajustar ao sistema e à organização. São atividades da modelagem de negócios: definir os objetivos da modelagem de negócios, avaliar a condição atual do negócio com a intenção de refinar os objetivos, capturar um vocabulário comum, explorar as partes do negócio que podem ser automatizadas, desenvolver e estruturar um modelo de casos de uso[3] para demonstrar uma visão geral da funcionalidade do sistema e se necessário desenvolver também um modelo de domínio[4] e finalmente identificar metas e processos de negócios. (MOREIRA, 2011)

[3] O modelo de casos de uso é um modelo das funções pretendidas do sistema e seu ambiente, e serve como um contrato estabelecido entre o cliente e os desenvolvedores. O modelo de casos de uso é usado como fonte de informações essencial para atividades de análise, design e teste.
[4] É a representação visual das classes do sistema ou do problema proposto, é utilizado para projetar o software.

Os Requisitos são estabelecidos de acordo com as necessidades dos clientes. De acordo com o que o sistema deve fazer. Devem fornecer uma base para planejar o conteúdo do sistema. Nesta fase é que são definidas as interfaces e delimitações do sistema. São atividades relativas aos requisitos: analisar o problema e propor soluções, estimar tempo e custos, compreender as funcionalidades do sistema, gerenciar requisitos da iteração[5]. (MOREIRA, 2011)

A Análise e Projeto mapeia os requisitos em um projeto detalhado de como o sistema deve ser criado, adapta o projeto para a satisfação dos requisitos, aperfeiçoa os resultados da análise, seleciona as melhores estratégias para implementar as soluções. São atividades da Análise e Projeto: realizar síntese arquitetural, para demonstrar que o sistema é factível, identificar serviços candidatos e também recursos já existentes na organização, esse último com a intenção de preservá-los ao máximo, analisar comportamentos que os requisitos possam trazer, projetar estruturas de banco de dados e componentes do sistema, especificação estrutural e comportamental de serviço. (MOREIRA, 2011)

A Implementação corresponde à elaboração e preparação dos módulos necessários à execução do software. Define a organização de códigos, divide os subsistemas. Testa os componentes separadamente. São atividades relacionadas à implementação: definir como serão feitas a implementação e a integração, implementar classes e objetos em termos de componentes, integrar todos os resultados produzidos (subsistemas) pela equipe em um sistema executável, integrar o sistema e integrar todos os builds gerados por todos os subsistemas. (MOREIRA, 2011)

A disciplina de Testes é a investigação dos códigos, a fim de encontrar defeitos, erros e falhas. Nesta disciplina são validadas as funcionalidades do sistema e são observados os aspectos de qualidade em relação ao contexto. Verificam-se as questões de funcionalidade, segurança, usabilidade, desempenho, suportabilidade, aceitabilidade entre outros. (MOREIRA, 2011)

Segundo Moreira (2011) a disciplina de testes é composta dos seguintes elementos:

a) **Planos de testes:** onde o contexto do teste é planejado. Eles servem para direcionar, orientar e restringir o esforço de teste. Pode-se ter um plano de testes para cada iteração. Descreve os tipos de testes que serão utilizados. Os

[5] Abrange as atividades do desenvolvimento de software em curtos períodos que resulta em um release de um produto executável.

planos variam de organização pra organização, em algumas podem ser informais e em outras, altamente formalizados.

b) **Desenvolvimento:** Casos de testes refletem os requisitos que devem ser verificados. Se ele verifica que um requisito foi atendido, o caso de teste deu positivo. Constituem a base dos scripts de testes (instruções passo a passo que permitem a execução de um teste). Os dados de testes são os utilizados para a realização dos testes e o cenário de testes descreve uma técnica para testar o sistema do ponto de vista do usuário.

c) **Execução:** Os casos de teste são executados e é feita uma avaliação dos resultados dos testes.

d) **Níveis de testes:** São realizados testes em vários estágios, é mais rápido e eficiente testar as partes menores primeiro para depois testar a integração delas. Os estágios são: testes de unidade, que testa cada módulo isoladamente, seja ele uma função ou uma classe, este teste é feito pelo desenvolvedor; teste de integração, que testa os módulos e subsistemas juntos, também é feito pelos desenvolvedores; teste de sistema, que simula o ambiente de produção, feito pelos desenvolvedores e por uma equipe independente; e finalmente os testes de aceitação, que é a última ação de testes antes da implementação do software, ele verifica o sistema em relação aos seus requisitos originais e é realizado pelos usuários finais.

e) **Visões de testes:** Podem ser testes de caixa preta (visão do usuário), onde são testadas somente as interfaces do sistema ignorando-se a parte interna do software e avalia os dados de entrada e saída até o limite da aceitabilidade; ou de caixa branca (visão dos desenvolvedores), que são feitos somente na parte interna, se encarregam de garantir que todos os comandos do programa tenham sido testados.

As atividades relacionadas aos testes são: estabelecer metas que conduzirão aos testes, avaliar se os testes atingem seus propósitos, aprimorar recursos de testes. (MOREIRA, 2011)

A disciplina de Implantação é feita à medida que o sistema vai ficando pronto. O conjunto de todas as unidades de implantação forma o produto final. Nesta disciplina todos os arquivos executáveis são testados e o produto é empacotado e distribuído. Para isto, também é definida a lista de materiais que compõem o produto como: documentos que indicam como o produto deve ser instalado, scripts, mídias, alterações incorporadas na versão entre outras. A

cada build a lista de materiais é atualizada juntamente com os erros e recursos problemáticos. (MOREIRA, 2011)

As atividades relacionadas à implantação são: planejar a implantação, desenvolver materiais de treinamento e suporte aos usuários, gerenciar testes e planos de aceitação (Descreve como o cliente avalia o produto e se a lista de materiais é aceitável pelo usuário), produzir a Unidade de Implantação - executáveis de instalação de software, liberar o sistema para testes, empacotar produto, disponibilizar o sistema, estabelecer banco de dados de relatos de problemas, fornecer assistência para correção de erros. (MOREIRA, 2011)

A disciplina de Gestão de Configuração e Mudanças (GCM), segundo Moreira (2011), avalia os custos e programa o impacto das mudanças sobre o produto. Trata da definição de configurações, da criação e identificação de versões. Utiliza a técnica da linha base, que é o conjunto de itens que compõem uma versão do software. Os softwares de controle de versões ajudam a gerir a linha Base. O controle de versões registra toda a evolução do projeto, cada alteração de cada arquivo. Permite que vários desenvolvedores trabalhem em conjunto nos mesmos arquivos, sem sobrescrever códigos de outros.

A GCM tem por objetivo assegurar a integralidade e correção do produto, restringir alterações de acordo com os requisitos, fornecer auditoria para saber quem fez as mudanças e se podemos reproduzir essa mudança.

As atividades relacionadas à GCM são planejar o controle das mudanças e versões, gerenciar solicitações de mudanças – avaliar o custo e o impacto da mudança, garantir a precisão, integridade e consistência do produto, criar e configurar ambiente para o projeto (repositório do projeto onde ficam todas as pastas, arquivos, *plugins* e dados do software), criar espaço de trabalho de cada desenvolvedor.

A Gestão de Projetos fornece estrutura para gerenciar, planejar, formar equipe, executar e monitorar projetos. O RUP não gerencia pessoas (contratação, treinamento), contratos e orçamentos de clientes.

As atividades relacionadas à gestão de projetos são: planejar e conceber o projeto avaliando os riscos e o escopo – atualizar o caso de negócio[6], planejar e gerenciar iterações, monitorar e programar os trabalhos, controlar status do projeto, aprimorar o plano de desenvolvimento, fechamento tanto das iterações quanto do projeto assegurando que os objetivos foram atingidos. O plano de desenvolvimento de software reúne todas as informações necessárias ao gerenciamento do projeto, contém plano de gestão de riscos,

[6] Fornece informações do negócio para verificar seus objetivos, viabilidade, previsão financeira, restrições.

resolução de problemas, aceitação do produto, garantia de qualidade entre outros. (MOREIRA, 2011)

A disciplina de Ambiente é focada nas atividades relacionadas à adaptação do processo organizacional. Provem a organização do desenvolvimento e todos os elementos que darão suporte à equipe como: estrutura, processos e ferramentas. Ela também fornece todos os ambientes necessários ao desenvolvimento, exemplos: ambiente de desenvolvimento, de testes e de produção.

As atividades relacionadas ao ambiente são: seleção, aquisição e instalação de ferramentas, configuração e melhoria de processos, suporte técnico, preparar os ambientes que darão apoio ao desenvolvimento. (MOREIRA, 2011)

2.3 XP (*Extreme Programming*)

Em meados de 1990, Kent Beck procurou formas mais simples e eficientes de desenvolver software, em março de 1996 ele iniciou um projeto e propôs uma metodologia inovadora chamada XP - *Extreme Programming*, ou seja, Programação Extrema.

XP é talvez o mais conhecido e utilizado dos métodos ágeis. Utiliza boas práticas como: desenvolvimento iterativo, envolvimento do cliente como parte da equipe, refatoração entre outros. (SOMMERVILLE, 2011)

O nome *Extreme Programming* se dá por levar ao extremo seus princípios e práticas como: testes, programação em par, iterações muito pequenas entre outros.

Estilos de codificação são decididos em grupo, pois a XP valoriza muito o trabalho em equipe, o respeito entre os desenvolvedores, a simplicidade perante o desenvolvimento, com a intenção de se fazer apenas o necessário primeiro, evitando vir a fazer o desnecessário gastando assim menos tempo e dinheiro.

A XP prioriza a comunicação e ela é feita por diálogos presenciais constantes, estabelecendo pontos importantes do projeto de software, e dando atenção a todos os detalhes como gestos, expressões faciais, postura, tom de voz, entre outros, onde o cliente expõe suas necessidades e a equipe estima custos e prazos. Dando oportunidade de compreensão do projeto a todos os envolvidos e permitindo o melhor planejamento. (COSTA, 2011)

A equipe de desenvolvimento lida com as mudanças de maneira natural, busca se adaptar as mudanças com coragem e segurança, confiando nos mecanismos de proteção. (COSTA, 2011)

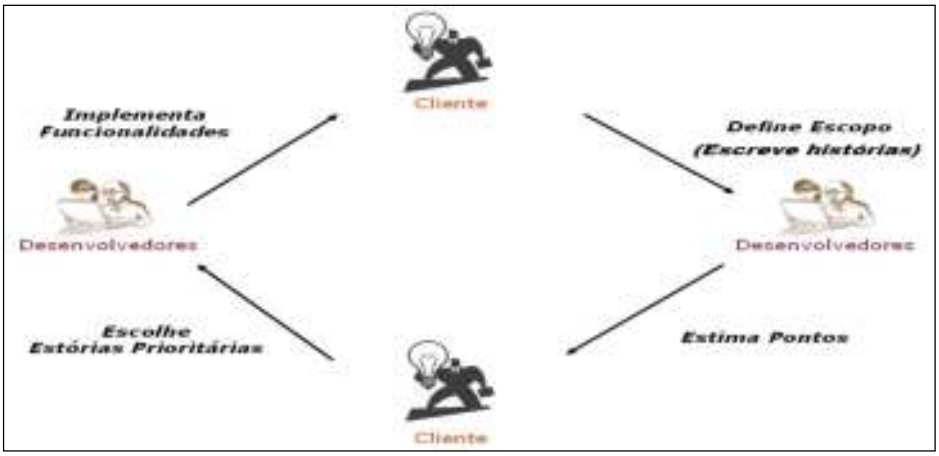

Figura 3 - Ciclo de vida de uma estória em projeto XP
Fonte: MEDEIROS (2011)

É sabido que qualquer projeto esta sujeito a mudanças, falhas e problemas, por isso os desenvolvedores tentam diminuir ao máximo o tempo gasto com cada funcionalidade, a fim de descobrir os problemas mais cedo. Assim o cliente logo tem contato com cada funcionalidade podendo testar e verificar o que ainda falta, o que pode ser mudado ou excluído.

O método é voltado para orientação a objetos, equipes pequenas ou médias com até 12 desenvolvedores de preferência e requisitos que podem mudar frequentemente.

2.3.1 Fases da XP

Como mostrado na Figura 3, o planejamento começa com a criação de um conjunto de estórias feitas pelos usuários que descrevem requisitos e necessidades do sistema a ser construído. Membros da XP avaliam as estórias e lhes atribuem custos e prazos. O desenvolvedor escolhe primeiro as que têm maior prioridade. Cada incremento do software é uma versão contendo o acordo das estórias, data de entrega e outros assuntos do projeto.

> Mudanças nos requisitos são solicitadas informalmente. Como consequência, o âmbito do projeto pode mudar e trabalhos anteriores podem ter que ser modificados para acomodar as necessidades atuais. Os defensores argumentam que isso acontece independentemente do processo que é aplicado e que a XP oferece mecanismos para controlar a extensão do escopo. (PRESSMAN 2010)

A XP valoriza projetos simples, da maneira que foi descrita na estória. Ela também encoraja o uso de cartões CRC (*Class-Responsability-Colaborator*) descrevendo as classes com suas responsabilidades e colaboradores, como mostrado na Figura 4. São utilizados na etapa inicial e é feito por uma equipe de cinco ou seis pessoas como: o usuário, o analista, programador entre outros.

Classe: **Pedido**	
Responsabilidade	Colaboração
Conhecer data do pedido	**Colaborador um**
Conhecer data da entrega	**Colaborador dois**
Conhecer número do pedido	**Colaborador três**

Figura 4: Exemplo de CRC

Fonte: Adaptada de BEZERRA (2011)

Se um problema difícil é encontrado recomenda-se a criação de um protótipo operacional, com a intenção de validar a estimativa inicial da estória que contém o problema. O projeto vai sendo modificado à medida que o desenvolvimento avança.

Outra característica de XP é a refatoração, que procura possíveis melhorias para o software, remove código duplicado, simplifica processos, melhorando a compreensão dos códigos. (SOMMERVILLE, 2011)

Antes de iniciar a Codificação ocorrem os testes unitários que exercitam cada estória, cada método, assim o desenvolvedor fica mais bem preparado para criar os códigos.

A XP recomenda a programação em pares, duas pessoas trabalham em um mesmo computador, onde cada pessoa assume um papel ligeiramente diferente, garantindo melhor qualidade ao desenvolvimento.

À medida que os pares vão desenvolvendo, outra equipe de desenvolvedores se encarrega de integrar os códigos ou até mesmo o próprio par usando a estratégia de integração contínua evitando problemas de compatibilidade.

A parte de testes da XP começa mesmo antes da codificação, ajuda a solidificar as exigências melhorando o desenvolvimento.

Todos os códigos devem ter testes unitários (*unit tests*), sem isso ele não pode ser entregue, pois ele garante um melhor nível de desempenho. Os testes de validação e integração podem ocorrer diariamente. Testes de aceitação podem ser escritos pelo cliente ou pela equipe de testes com a finalidade de testar todo o sistema.

2.4. *Agile Software Developmentt* (ou Desenvolvimento ágil de software)

Para dar apoio à modelagem de negócios, a XP e o RUP podem utilizar desta metodologia de modelagem ágil, o Agile.

Em 2001 dezessete membros da comunidade mundial de desenvolvimento entre eles, Robert C. Martin, Jim Highsmith, Kent Beck e outros, reuniram-se para discutir as boas práticas que cada um utilizava em seus trabalhos. A partir daí criaram um conjunto de práticas para desenvolver e ajudar outros a desenvolverem softwares.

As conclusões foram publicadas no "Manifesto para Desenvolvimento Ágil de Software", que é uma organização não lucrativa com a finalidade de promover desenvolvimento ágil. (BECK et al, 2011)

Através do Manifesto Ágil, conforme Beck et al (2011), passou-se a valorizar mais os indivíduos e a iteração entre eles mais que processos e ferramentas; o software em funcionamento mais que documentação abrangente; a colaboração com o cliente mais que negociação de contratos e a responder a mudanças mais que seguir um plano. Ambler (2004) define a modelagem ágil como:

> uma metodologia baseada na prática para modelagem e documentação eficazes de sistemas baseados em software. É um conjunto de práticas guiado por princípios e valores para profissionais de software aplicarem em seu dia a dia. Em outras palavras, não define procedimentos detalhados sobre como criar um determinado tipo de modelo. Em vez disso, fornece conselhos sobre como ser um modelador eficiente.

Segundo Beck et al (2011) os doze princípios do Manifesto Ágil são:

- Maior prioridade: consiste em satisfazer o cliente, através da entrega adiantada e contínua de software;
- Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adéquam a mudanças, para que o cliente possa tirar vantagens competitivas;
- Entregar partes do software funcionando com freqüência, na escala de semanas até meses, com preferência aos períodos mais curtos;
- Pessoas relacionadas a negócios e desenvolvedores devem trabalhar em conjunto e diariamente, durante todo o curso do projeto;

- Construir projetos ao redor de indivíduos motivados. Dando a eles o ambiente e suporte necessário, e confiar que farão seu trabalho;
- O método mais eficiente e eficaz de transmitir informações para, e por dentro de um time de desenvolvimento, é através de uma conversa cara a cara;
- Software funcional é a medida primária de progresso;
- Processos ágeis promovem um ambiente sustentável. Os patrocinadores, desenvolvedores e usuários, devem ser capazes de manter indefinidamente, passos constantes;
- Contínua atenção a excelência técnica e bom design aumentam a agilidade;
- Simplicidade: a arte de maximizar a quantidade de trabalho que não precisou ser feito.
- As melhores arquiteturas, requisitos e designs emergem de times auto-organizáveis;
- Em intervalos regulares, o time reflete em como ficar mais efetivo, então, se ajustam e otimizam seu comportamento de acordo.

As iterações são normalmente curtas entre uma e quatro semanas. Cada iteração inclui: planejamento e análise de requisitos, projeto rápido, geração de código, testes unitários e testes de aceitação quando um produto funcional é entregue ao cliente.

O desenvolvimento ágil da ênfase na facilitação da geração rápida de softwares é menos orientado a documentos e mais orientado a códigos. Métodos ágeis são mais adequados a equipes pequenas com no máximo de 20 a 40 pessoas.

3. Comparações e análises entre as metodologias

Nos próximos tópicos serão apresentadas as análises de: comparação e viabilidade entre as metodologias com base na teoria aqui sintetizada.

3.1. Análise comparativa

Depois de analisar individualmente cada metodologia, vamos analisar neste capítulo seus pontos marcantes e no final um resumo em forma de tabela da comparação entre elas.

a) Cascata: O modelo Cascata funciona bem quando os requisitos não mudam, são fixos e as atividades são feitas seqüencialmente, de forma linear. Porém, comparado aos outros modelos

aqui citados fica em desvantagens nos seguintes aspectos: não permite a participação de clientes na fase de desenvolvimento, o que impossibilita evitar problemas; não é flexível no que se refere à correção de falhas; a estimativa de prazo de entrega é imprecisa, pois o planejamento quase sempre precisa ser revisto; desenvolvimento burocrático e lento. Consequentemente a mitigação dos riscos do projeto ocorre tardiamente elevando os custos de mudanças.

b) RUP: De todas as metodologias aqui citadas RUP é a que melhor se encaixa em projetos diversificados, devido a suas amplas customizações. Segundo Vieira (2012) o RUP possui: foco na arquitetura, em equipes grandes, maior esforço na análise e orientação de modelagem por casos de uso.

Em contraposição ao XP, RUP tem forte documentação, o que para o RUP auxilia no desenvolvimento. Entretanto, como pouca gente se dedica realmente a adequar o RUP às necessidades de cada projeto, na prática, muitas pessoas acabam utilizando os modelos de documentos como proposto pelo RUP, o que é um erro, pois os modelos foram concebidos para os maiores projetos possíveis e deveriam ser adequados ao tamanho de cada projeto. Isto pode levar algumas pessoas a concluir que o RUP é burocrático, o que não é necessariamente verdade, pois ele pode estar sendo mal utilizado. O uso de documentação, tal qual é proposta nos modelos do RUP, ao mesmo tempo pode atrapalhar quando houver mudanças no projeto, pois além da mudança nos códigos tem de haver mudança também na documentação que já foi feita.

c) XP: O modelo XP enfatiza a rápida entrega de software, flexibilidade e que os clientes são partes da equipe. Os sistemas desenvolvidos são capazes de se adequar a modificações, são preparados para isso. Ainda segundo Vieira (2012), a XP é restrita a equipes locais, possui: foco maior na validação, equipes pequenas, maior esforço em produzir código, orientação de modelagem baseada nas histórias. Em relação aos outros métodos XP produz pouca documentação e planejamento o que pode ser seu ponto fraco.

RUP e XP têm uma visão semelhante sobre as melhores práticas de desenvolvimento. Exemplos: desenvolvimento iterativo e foco no usuário final. (RUP 2012)

Todos os métodos de desenvolvimento abordados aqui têm em comum o Ciclo de Desenvolvimento/Atividades:

Na XP e RUP o projeto é desenvolvido de forma iterativa (ao final de cada iteração tem se uma nova versão do sistema) e no modelo Cascata de forma sequencial.

No modelo Cascata uma fase só começa quando a outra tiver terminado, pois uma depende da outra. Dessa forma, uma versão do software pode demorar muito.

No RUP cada fase é quebrada em iterações onde são seguidas disciplinas, cada qual com seu conjunto de atividades relacionadas ao mesmo contexto e que ocorrem com um certo grau de paralelismo. As iterações são curtas e cada uma termina com a liberação de um produto entregável. O ciclo de desenvolvimento termina com uma versão completa e executável do produto. Nas iterações de Elaboração e especialmente de Construção, o ritmo de entregas é bem parecido com o do XP gerando uma versão com algum grau de exequibilidade a cada iteração.

Na XP as atividades são desenvolvidas simultaneamente. Pequenos releases (versões do software) são entregues em prazos curtos.

A Figura 4 resume, do ponto de vista de disciplinas (áreas do conhecimento da engenharia de software), os itens discutidos nesta seção. É comum alguns desenvolvedores usarem mais de uma metodologia ao mesmo tempo para buscar a melhor performance, exemplo: o uso das atividades de uma metodologia na iteração da outra fazendo com que haja uma espécie de metodologia híbrida entre elas.

Disciplina	Cascata	RUP	XP
Modelagem de negócios		X	X
Levantamento de requisitos	X	X	
Análise	X	X	
Projeto	X	X	X
Codificação/ Implementação	X	X	X
Testes	X	X	X
Implantação (Entrega)	X	X	
Gestão de configuração e mudanças	X	X	
Gestão de projeto		X	X
Ambiente		X	

Figura 4: Comparações entre as metodologias.

Fonte: adaptada de MOREIRA (2011)

3.2. Análise de Viabilidade

Metodologias de desenvolvimento buscam reduzir riscos e produzir software de qualidade, porém buscam de formas diferentes. Além de cada uma se aplicar melhor em determinados cenários.

De modo geral, o modelo em cascata deixou de ser recomendado pelo fato dos testes e a integração serem feitos apenas no final de cada fase, trazendo despesas caso os requisitos mudem, portanto o modelo em cascata deve ser utilizado somente quando os requisitos não forem sofrer alterações (o que é bem difícil de saber) e forem bem compreendidos.

A XP é recomendada em equipes pequenas ou médias e locais (não distribuída geograficamente) pelo fato das reuniões serem feitas presencialmente. Ela é indicada também pra projetos com requisitos voláteis, flexíveis e sem ênfase na documentação. Possui pouco foco na arquitetura, portanto acontecem mais erros e apesar de estarem aptos a se adaptarem as mudanças pode fazer com que o prazo se estenda.

O RUP distribui o projeto de forma bem definida, devido a sua estruturação. Logo, ele pode ser usado em projetos pequenos, médios ou grandes e com distribuição geográfica. Por ele ser mais bem planejado, geralmente os códigos ficam mais enxutos, com menor índice de erros.

A Figura 5 resume o conteúdo discutido nesta seção. Quanto à melhor metodologia, não é possível fazer uma conclusão genérica, a escolha depende essencialmente do quanto se conhece dos requisitos, da distribuição geográfica dos clientes e usuários chaves, da complexidade técnica (que deve ser vista como forma de determinar o nível de formalismo necessário) e do tamanho do projeto. Geralmente para projetos grandes, o mais aconselhável é o RUP, e em projetos menores a XP pode ser mais recomendada por produzirem de maneira mais rápida, mesmo que não tão organizada.

Características	**Cascata**	**RUP**	**XP**
Adaptação a mudanças	Retrabalho	Se auto organiza para lidar com as mudanças	Rápida
Comunicação *Documentação*	Comunicação no início e fim Especificação do sistema	Geralmente formal Artefatos	Orais, histórias dos clientes Documentação fraca
Ferramentas	Várias	IBM	Não específica (sugestão: Xplanner e Junit)
Planejamento	Bem definido	Análise detalhada seguindo a prática do planejamento em ondas	Menos planejamento, mais esforço em produzir o software
Magnitude dos projetos	Qualquer tamanho, mas com requisitos estáveis	Pequenos a grandes projetos e distribuídos	Pequenos a médios (locais)
Entrega	Perto do fim	A cada iteração	Versões são entregues a cada release

Figura 5: Análise de viabilidade.

Fonte: adaptada de NÓBREGA JÚNIOR (2011)

A Figura 6 resume a visão de aplicabilidade de cada método estudado, utilizando como critérios a complexidade de gestão e técnica, que determinam indiretamente o nível de formalidade (informal ou formal) e de integração (cascata ou iterativo) necessários.

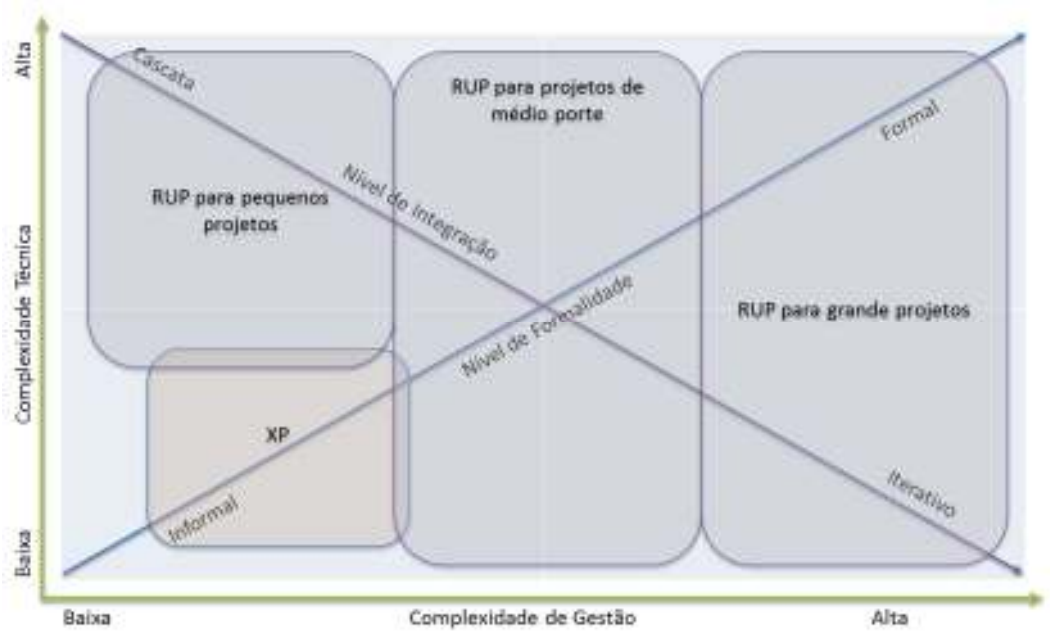

Figura 6: Aplicabilidade dos métodos

Fonte: Adaptada de MOREIRA (2011)

4. Considerações Finais

Neste artigo foram apresentadas três metodologias de desenvolvimento de software. Foi feita uma pesquisa bibliográfica e uma explanação de cada metodologia, a comparação entre elas e a análise de viabilidade.

O objetivo do artigo foi mostrar qual a melhor metodologia de desenvolvimento a ser usada. Chegou-se a conclusão que o modelo Cascata é o menos indicado, pois por ser desenvolvido de forma sequencial e deixar testes e integração para o final de cada fase, traz despesas caso os requisitos mudem, seu uso é indicado quando os requisitos não sofrem alterações e forem bem compreendidos, o que limita bastante a utilização deste método.

O RUP e a XP são indicados por conseguirem atingir o objetivo a que se destina que é diminuir os riscos, entregar software de qualidade e na data prevista. Foi constatado que o uso deles depende do tamanho, da necessidade do projeto, dentre outros fatores.

O RUP devido a sua ampla customização pode ser adaptado para projetos de qualquer escala e com distribuição geográfica. Como os testes são feitos em todas as iterações, fica

mais fácil encontrar os erros e por serem iterações pequenas é mais fácil acomodar as mudanças.

Já a XP é recomendada em equipes pequenas ou médias e locais pelo fato das reuniões serem feitas presencialmente. É adaptável a mudanças e possui pouco foco na arquitetura, porém, com isso pode ocorrer mais erros e apesar de estarem aptos a se adaptarem as mudanças pode fazer com que o prazo se estenda.

5. Referências

AMBLER, Scott. *Modelagem Ágil:* Práticas eficazes para a Programação eXtrema e o Processo Unificado. Bookman. 2004.

BECK, K. et al. *Manifesto ágil*. Disponível em: <http://manifestoagil.com.br/>. Acesso em: 18 out. 2011.

BEZERRA, Eduardo. *Princípios de Análise e Projeto de Sistemas com UML*. Disponível em: <http://pt.scribd.com/lperes_5/d/50470540/47-Exemplos-de-cartoes-CRC>. Acesso em: 20 out. 2011.

CARLOS, A. et al. *Engenharia de Software - Modelo Cascata*. Disponível em: <http://www.slideshare.net/erysonsi/modelo-cascata-apresentao-4345883>. Acesso em: 17 out. 2011.

COSTA, André. *Metodologia Ágil de Desenvolvimento.* Disponível em: <http://www.andrezip.com.br/metodologia-agil-de-desenvolvimento-de-software-extreme-programming-xp/>. Acesso em: 05 out. 2011.

MEDEIROS, Manoel Pimentel. *Planejando seu projeto com eXtreme Programming – Parte I*. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=4273>. Acesso em 08 ago. 2011.

MOREIRA, Márcio. *Metodologia de Desenvolvimento de Software – RUP*. Pitágoras. 2011. Disponível em: <http://si.lopesgazzani.com.br/docentes/marcio/rup/index.htm>. Acesso em: 07 dez. 2011.

NÓBREGA JÚNIOR, Nelson Alves. *A Arquitetura no processo de desenvolvimento de software.* Disponível em: < http://sites.google.com/site/nelsonjuniorbr2/as-20091 >. Acesso em 18 out. 2011.

PRESSMAN, Roger S. *Engenharia de software.* New York: McGraw-Hill, 2010.

SOMMERVILLE, Ian. Engenharia de Software. Massachusetts: Boston: Addison-Wesley, 2011.

RUP, Práticas Dinâmicas no. Disponível em: <http://www.wthreex.com/rup/tour/rm_xp2rup.htm#XPPractices>. Acesso em: 13 jan. 2012.

VASCO, Carlos G.; VITHOFT, Marcelo Henrique; ESTANTE, Paulo Roberto C. *Comparação entre Metodologias RUP e XP*. Disponível em: <http://www.edilms.eti.br/uploads/file/bd/RUPvsXP_draft.pdf>. Acesso em: 18 out. 2011.

VIEIRA, Felipe José Rocha. *Análise comparativa entre o RUP e a XP*. Universidade Federal de Sergipe. Disponível em: <http://www.google.com.br/url?sa=t&rct=j&q=compara%C3%A7%C3%B5es%20rup%2C%20xp%20e%20cascata&source=web&cd=10&ved=0CGcQFjAJ&url=http%3A%2F%2Fitatechjr.com.br%2Fsecomp%2Fdownloads%2Fmateriais-palestras-e-mc%2Fdoc_download%2F35-analise-rup-xp.html&ei=MS8nT9S8Cc36gge5saGHCQ&usg=AFQjCNE-TyLmEocHMg20QLxXB3Q-y1HyNw&cad=rja>. Acesso em: 13 jan. 2012.

VIGGIANO, Eros. *Modelagem Arquitetural com o MS Visio*. Disponível em: <http://blog.arkhi.com.br/page/5/>. Acesso em: 20 out. 2011.