INSTITUTO DE ENSINO SUPERIOR DO ESPÍRITO SANTO
FACULDADE DO ESPÍRITO SANTO
SISTEMAS DE INFORMAÇÃO

EDUARDO ROSA DA SILVA
LUCIANO SILVA FERREIRA

GERENCIAMENTO DE PROJETOS APLICADO NO DESENVOLVIMENTO DE SOFTWARES EDUCATIVOS

CACHOEIRO DE ITAPEMIRIM – ES
2013

EDUARDO ROSA DA SILVA
LUCIANO SILVA FERREIRA

GERENCIAMENTO DE PROJETOS APLICADO NO DESENVOLVIMENTO DE SOFTWARES EDUCATIVOS

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação na Faculdade do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Alexandre Romanelli

CACHOEIRO DE ITAPEMIRIM – ES
2013

EDUARDO ROSA DA SILVA
LUCIANO SILVA FERREIRA

GERENCIAMENTO DE PROJETOS APLICADO NO DESENVOLVIMENTO DE SOFTWARES EDUCATIVOS

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação na Faculdade do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Aprovado em 05 de Novembro de 2013.

COMISSÃO EXAMINADORA

_____
Prof. Me. Alexandre Romanelli
Orientador

_____
Prof. Me. Jocimar Fernandes

_____
Prof. Me. Antonio Carlos Andrade

Dedicamos aos nossos familiares, professores e colegas de faculdade.

AGRADECIMENTOS

Agradecemos a Deus, que iluminou nosso caminho durante esta caminhada.
Agradecemos a nossos familiares, que foram a base para que chegássemos a esta etapa tão importante de nossas vidas.
Agradecemos aos professores que nos acompanharam durante toda a graduação, em especial ao professor Alexandre Romanelli, responsável por nos orientar na elaboração deste trabalho.
Agradecemos aos amigos e colegas, pelo incentivo e pelo apoio constantes.

"A vida acadêmica é essencial para nos mostrar a base teórica da profissão que queremos seguir, mas somente na prática do dia-a-dia é que colocaremos a prova tudo aquilo que os mestres nos ensinaram e muitas vezes veremos que não era bem daquela maneira."
Luís Alves

"*A vida é um longo caminho que começamos a trilhar desde o primeiro dia, no começo engatinhando, depois caminhando passo a passo até conquistarmos a confiança necessária para correr, sim, correr, pois o caminho é longo e o sucesso não espera.****"***
Luís Alves

SILVA, Eduardo Rosa da; FERREIRA, Luciano Silva. **Introdução ao gerenciamento de projetos aplicado no desenvolvimento de softwares educativos**. 2013. Trabalho de Conclusão de Curso (Sistemas de Informação) – Faculdade do Espírito Santo, Cachoeiro de Itapemirim, 2013.

RESUMO

Na atualidade, grande parte das empresas que trabalham com desenvolvimento de software estão procurando fazer uma boa gerência de seus projetos, investindo na otimização de processos fazendo com que os requisitos exigidos pelo cliente sejam atendidos e se obtenha qualidade no produto final. Esta pesquisa tem como objetivo principal mostrar a importância do gerenciamento de projetos para haver organização no planejamento e execução das atividades que envolvem o desenvolvimento de softwares educativos, mostrando como estes podem auxiliar no aperfeiçoamento e na complementação da qualidade de ensino das escolas. Além disso, serão abordadas informações sobre metodologias que podem ser usadas por equipes de desenvolvimento de software educativo no auxílio da gerência de seus projetos, mostrando ainda que a escolha de uma delas deve ser baseada em alguns fatores como o perfil da equipe, o tamanho do projeto e os objetivos a serem alcançados.

Palavras-chave: Gerenciamento. Projetos. RUP. Scrum. Software.

SILVA, Eduardo Rosa da; FERREIRA, Luciano Silva. **Introdução ao gerenciamento de projetos aplicado no desenvolvimento de softwares educativos**. 2013. Trabalho de Conclusão de Curso (Sistemas de Informação) – Faculdade do Espírito Santo, Cachoeiro de Itapemirim, 2013.

ABSTRACT

In actuality, most of the companies that work with software development are looking to make a good management of your projects, investing in optimizing processes so that customer requirements are met and if get quality in the final product. This research has as main objective to show the importance of project management to be organization in the planning and execution of activities involving the development of educational software, showing how these can aid in the improvement and in the quality of education complementation schools. In addition, information will be addressed on methodologies that can be used by development teams of educational software in your management aid projects, showing that the choice of one of them must be based on some factors such as the profile of the team, the size of the project and the goals to be achieved.

Key-words: Management. Projects. RUP. Scrum. Software.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ANSI - American National Standards Institute (Instituto Nacional Americano de Padrões).

API - Atendimento Personalizado em Imóveis.

CDS - Condomínio de Soluções Corporativas.

CMMI - Capability Maturity Model Integration (Modelo de Integração de Capacidade e Maturidade).

EUA - Estados Unidos da América.

IBM - International Business Machines (Máquinas de Negócio Internacionais).

ISD - Integrated System Diagnostics (Diagnóstico de Sistema Integrado).

ISO - International Organization for Standardization (Organização Internacional para Padronização).

JSD - Jackson System Development (Desenvolvimento de Sistema de Jackson).

MDD - Method Definition Document (Documento de Definição do Método).

PMBoK - Project Management Body of Knowledge (Área de Conhecimento do Gerenciamento de Projetos).

PMI - Project Management Institute (Instituto de Gerenciamento de Projetos).

PMP - Project Management Professional (Profissional de Gerenciamento de Projetos).

RUP - Rational Unified Process (Processo Unificado Rational).

SCAMPI - Standard CMMI Appraisal Method for Process Improvement (Padrão CMMI de Avaliação de Métodos para Melhoria de Processos).

SEI - Software Engineering Institute (Instituto de Engenharia de Software).

SWEBOK - Software Engineering Body of Knowledge (Área de Conhecimento da Engenharia de Software).

UML - Unified Modeling Language (Linguagem de Modelagem Unificada).

XP – Extreme Programming (Programação Extrema).

SUMÁRIO

1 INTRODUÇÃO

Empresas e profissionais que trabalham com desenvolvimento de softwares têm o conhecimento de que tal tarefa requer planejamento e execução de atividades, definidas conforme o que é estabelecido no projeto, sendo necessário lidar com variadas questões técnicas e gerenciais.

A gerência de projetos é extremamente importante no ambiente do desenvolvimento de softwares, pois através dela é possível auxiliar as equipes no planejamento e na correta execução das atividades que envolvem todo o ciclo de vida de desenvolvimento, o que irá gerar qualidade no produto final.

Além de avaliar a visão de diversos autores sobre o assunto, é importante também ser avaliado os modelos propostos por algumas das principais instituições na área, o SEI (Software Engineering Institute) e o PMI (Project Management Institute) que serão abordados durante a pesquisa.

1.1 Objetivos

Nas seções a seguir será abordado o objetivo geral e os objetivos específicos de nossa pesquisa.

1.1.1 Objetivo geral

Este trabalho tem como objetivo principal demonstrar a importância da utilização de técnicas de gerenciamento de projetos associadas com metodologias de desenvolvimento de softwares para o ambiente educacional, a fim de aprimorar a qualidade do ensino das escolas.

1.1.2 Objetivos específicos

Além do objetivo geral, nossa pesquisa tem alguns objetivos específicos:

- Aprofundar os estudos na área de gerenciamento de projetos;

- Constatar o quanto é importante a gerência de projetos em relação ao planejamento, execução e organização das atividades que envolvem o ciclo de vida da engenharia de software;

- Descrever argumentos que mostrem como os softwares educacionais podem auxiliar na qualidade de ensino nas escolas;

- Explicar as características de algumas das principais metodologias de gerência de projetos utilizadas pelas empresas que trabalham com desenvolvimento de software, sendo elas o RUP, Scrum, XP e ainda o guia prático do PMBOK;

1.2 Justificativa

A área de gerenciamento de projetos abrange vários processos que visam organizar o planejamento e a execução das atividades que envolvem o desenvolvimento de software. Para se criar softwares educativos com qualidade, é necessário haver rapidez e organização nas atividades que são proporcionadas com a ajuda da gerência de projetos, o que justifica a escolha deste tema para nossa pesquisa, pois os principais problemas enfrentados por organizações que trabalham com engenharia de softwares são originados pela falta ou péssima gerência dos projetos. Além do mais, é importante inserir nas escolas ferramentas tecnológicas que complementem e auxiliem os educadores na melhoria da qualidade do ensino.

1.3 Metodologia

A metodologia utilizada como fonte de pesquisa para elaboração deste trabalho são livros com os autores mais conceituados na área de engenharia de softwares e gerenciamento de projetos. Os três principais autores que embasam nossa pesquisa são Hélio Engholm Júnior, Shari Lawrence Pfleeger e Ian Sommerville.

Ademais, também estamos nos baseando em sites confiáveis que possuem artigos e informações nas áreas citadas, o que faz enriquecer as informações contidas em nossa pesquisa, a fim de demonstrar a importância do gerenciamento de projetos no ambiente de desenvolvimento de softwares educativos.

1.4 Organização do trabalho

Este trabalho está dividido em seis capítulos, incluindo este de introdução, que descreveu brevemente sobre o assunto, os objetivos, a justificativa e a metodologia aplicada.

O capítulo 2 descreverá a forma que alguns especialistas definem o desenvolvimento de software, além de citar o CMMI (modelo de qualidade que preza pela otimização dos processos que envolvem o ciclo de um projeto) e ainda abordar as principais informações sobre as áreas de conhecimento da engenharia de software, mostrando ainda a importância do gerenciamento de projetos em relação a estas áreas.

No capítulo 3 será mostrada a evolução das metodologias de desenvolvimento de software com o passar do tempo, abordando sobre os modelos pioneiros de ciclo de vida e suas principais características, assim como os modelos contemporâneos (os tradicionais e os motivos do surgimento das metodologias ágeis).

No capítulo 4 será descrito sobre os conceitos de gerenciamento de projetos e sua importância para o planejamento e execução das atividades que envolvem o ciclo de vida da engenharia de software.

O capítulo 5 abordará informações sobre a importância da utilização de softwares educativos como complementação e aprimoramento na qualidade de ensino das escolas, também mostrando a importância do uso de técnicas de gerenciamento de projetos associadas a metodologias de engenharia de software para haver qualidade nos softwares educativos desenvolvidos.

O capítulo 6 abordará sobre as conclusões obtidas com a realização desta pesquisa.

2 DESENVOLVIMENTO DE SOFTWARE

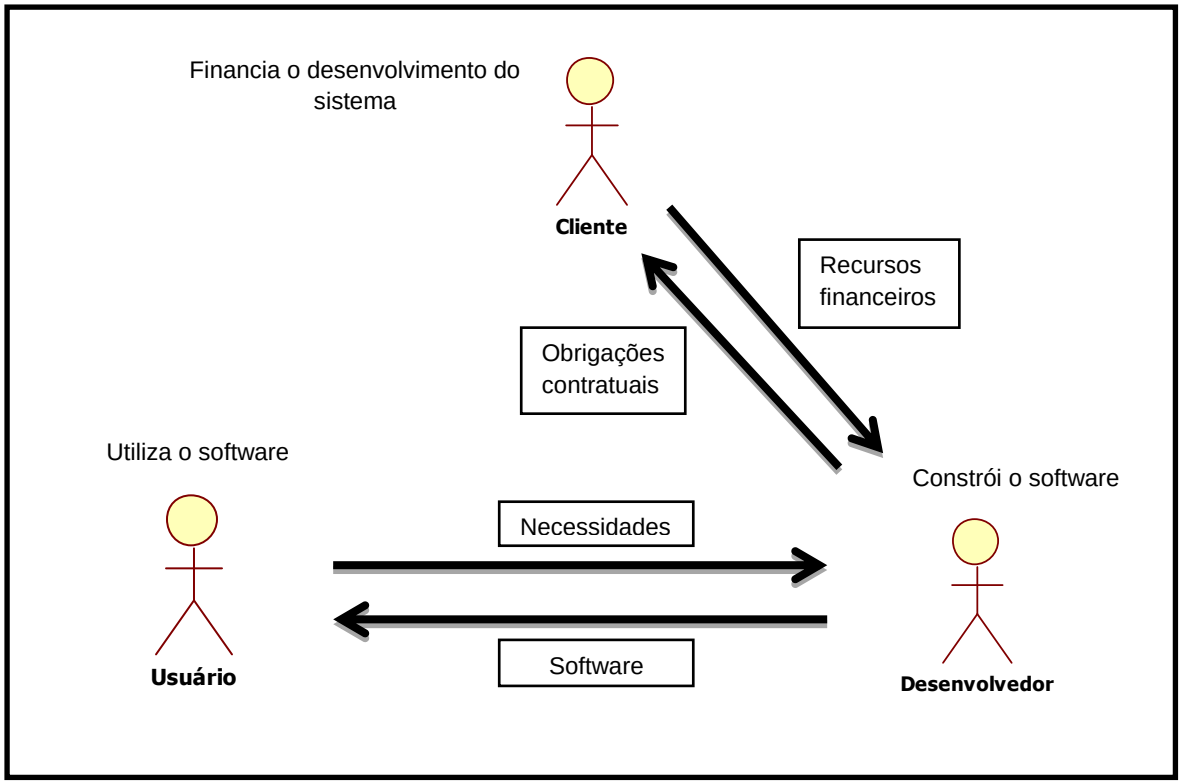
O desenvolvimento de software pode ser caracterizado como um conjunto de atividades necessárias para especificar, projetar e testar retorno dos resultados do software que está sendo criado (RAMOS, acesso em: 10 ago. 2013).

A utilização de softwares por empresas na realização dos mais variados serviços vem se tornando cada vez mais importante, tornando necessária a adoção de modelos e processos que visam à otimização da qualidade e diminuição dos custos no desenvolvimento e manutenção de sistemas. O gerenciamento de projetos nesta área tem extrema relevância, pois os softwares construídos visam atender as necessidades dos clientes. Isto só será obtido se houver organização no planejamento e execução das atividades para este fim, ao qual é a principal função da gerência de projetos.

> No desenvolvimento de software, podemos associar técnicas de gerenciamento de projeto com técnicas específicas de engenharia de software e utilizar processos definidos para desenvolver software, atingindo os requisitos funcionais e não funcionais [...]. Não parece razoável que utilizando uma excelente metodologia de desenvolvimento de software sem a utilização de boas práticas de gerenciamento de projeto podemos ter um projeto sem armadilhas e sem possibilidade de fracasso. Isso comprova que precisamos ter um conjunto de disciplinas de áreas distintas para termos sucesso em nossos projetos de desenvolvimento de software [...] (ENGHOLM JÚNIOR, 2010, p. 48).

Normalmente, os participantes de um projeto pertencem a uma destas categorias: clientes, usuários ou desenvolvedores. O **cliente** pode ser uma empresa, organização ou pessoa que paga para o software ser desenvolvido. O **desenvolvedor** pode ser uma empresa, organização ou pessoa que irá construir o software para o cliente. Nesta categoria ainda existe a necessidade de haver alguns gerentes para orientar e coordenar os programadores e a equipe que realiza os testes. O **usuário** é a pessoa, ou pessoas, que irão utilizar o sistema, aqueles que irão inserir dados ou ler resultados (PFLEEGER, 2004). A figura 1 ilustra os participantes envolvidos em um projeto de desenvolvimento de software.

Figura 1 – Participantes de um projeto de desenvolvimento de software



Fonte: Adaptado de Pfleeger, 2004

Existem alguns métodos para desenvolvimento de software que possuem como objetivo facilitar a produção de software de alta qualidade dentro de custos adequados. Métodos como Análise Estruturada e JSD foram desenvolvidos por volta da década de 70. Tais métodos tentaram identificar os componentes funcionais básicos de um sistema. Por volta das décadas de 80 e 90, os métodos orientados a funções foram suplementados por métodos orientados a objetos, propostos por alguns estudiosos da área. Essas diferentes abordagens foram agora integradas em uma única abordagem, criada de acordo com o UML (SOMMERVILLE, 2007).

Em um projeto de desenvolvimento de software existem diversos processos, que variam de acordo com a complexidade do que será desenvolvido. Podemos chamar de processos um conjunto de regras que permitem organizar um projeto em determinadas atividades. Estes processos têm como objetivo garantir a qualidade no desenvolvimento de software, de forma que os integrantes da equipe façam com que o projeto consiga a certificação CMMI por meio da avaliação chamada SCAMPI.

2.1 CMMI

O CMMI (Capability Maturity Model Integration) é um modelo de maturidade mantido pelo SEI (Instituto de Engenharia de Software) que serve para avaliar a qualidade dos processos de software, prezando pela melhoria contínua dos processos que envolvem o ciclo de um projeto, ajudando uma organização a se tornar mais eficiente.

Para Engholm Júnior (2010, p. 40), “o CMMI descreve o que deve ser feito em relação ao processo de desenvolvimento de software, com a finalidade de gerar qualidade. [...]”.

Em relação ao CMMI, Souza (2008, acesso em: 13 ago. 2013) nos diz que “[...] modelos de capacidade e maturidade atingido em seus níveis ajudam a prever o comportamento de um determinado processo diante do cenário ao qual o projeto se encontra [...]”.

2.2 SCAMPI

Os métodos de avaliação SCAMPI (Standard CMMI Appraisal Method for Process Improvement) podem ser definidos como métodos utilizados para avaliar organizações conforme os modelos CMMI. As regras que asseguram objetividade na classificação das avalições são definidas pelo documento SCAMPI MDD (Method Definition Document). Os métodos de avaliação Classe A, B, C são os que compõem a chamada família de avaliações SCAMPI (ISD BRASIL, acesso em: 12 ago. 2013).

2.3 Áreas de Conhecimento da Engenharia de Software

O gerenciamento de projetos, tema central desta pesquisa, é o principal aliado do Gerenciamento de Engenharia de Software, sendo uma das áreas de conhecimento mais importantes, pois contribui para que todas as outras ocorram de maneira planejada e organizada. Nesta seção será comentado brevemente sobre as áreas de conhecimento.

A Engenharia de Software foi divida em 10 áreas de conhecimento conforme consta no SWEBOK (*Software Engineering Body of Knowledge*), um guia de uso e aplicação das melhores práticas nesta área, desenvolvido com os conhecimentos que foram recolhidos durante quatro décadas e revisado por muitos profissionais das mais variadas nacionalidades (GOMEDE, acesso em: 28 ago. 2013). Estas áreas serão abordadas a seguir.

2.3.1 Requisitos de Software

Os requisitos de software são as necessidades e expectativas dos que estão envolvidos no projeto para desenvolver um produto que irá contribuir com a solução de algum problema. Existem os requisitos funcionais, que correspondem às funcionalidades ou ações que o software deve fornecer, normalmente podendo ser relacionados aos casos de uso da UML. Também existem os requisitos não funcionais, ou seja, requisitos que não se relacionam com as funcionalidades do sistema, mas com as características do software ou do ambiente dele. Alguns exemplos de requisitos não funcionais: usabilidade, desempenho, reusabilidade, entre outros (ENGHOLM JÚNIOR, 2010).

2.3.2 Design de Software

Engholm Júnior (2010) destaca que o propósito do design é criar uma solução técnica que satisfaça os requisitos do software, ou seja, o design de software transforma os resultados obtidos na análise de requisitos em um ou mais documentos capazes de serem interpretados pelos desenvolvedores. Esta área de conhecimento da engenharia de software pode ser compreendida como o processo de definição da arquitetura, dos componentes, módulos, interfaces e dados para determinado software com o objetivo de atender aos requisitos envolvidos.

2.3.3 Construção de Software

Segundo SWEBOK (2004, apud GOMEDE, 2010, acesso em: 05 out. 2013), a construção está relacionada com “implementação do software, verificação, testes de unidade, [...]. Esta área está envolvida com todas as áreas de conhecimento, porém,

está fortemente ligada às áreas de Design e Teste de Software". O processo de desenvolvimento ou construção de software geralmente abrange de maneira significativa estas duas áreas.

2.3.4 Teste de Software

Segundo SWEBOK (2004, apud GOMEDE, 2010, acesso em: 05 out. 2013) "Teste de software é uma atividade executada para avaliar a qualidade do produto, buscando identificar os defeitos e problemas existentes", a fim de corrigir os possíveis erros identificados antes da entrega do sistema ao cliente.

A seguir será descrito alguns dos tipos de teste de software existentes (COELHO, apud GOMEDE, acesso em: 05 out. 2013):

- **Teste funcional:** verifica condições válidas e inválidas, as regras de negócio, aspectos funcionais do sistema;
- **Teste de desempenho:** verifica o tempo de resposta e processamento com a utilização do software para configurações distintas;
- **Teste de segurança e controle de acesso:** verificam se funcionam corretamente os mecanismos de proteção de dados e acesso.

2.3.5 Manutenção de Software

Após a conclusão do desenvolvimento de um software, e já sendo devidamente instalado para utilização do cliente em seu ambiente real, qualquer trabalho realizado para alterar o sistema depois que estiver em operação, é considerado como manutenção do software. Porém, esta é diferente da manutenção de hardware, onde ocorre reparo e substituição de componentes com defeitos ou que não funcionam adequadamente, também exigindo uma manutenção periódica. Diferentemente do hardware, o software não se degrada nem requer manutenção por um período determinado. A manutenção de software ocorre quando é necessário fazer alguma modificação para corrigir defeitos, melhorar o desempenho do sistema ou adaptá-lo (PFLEEGER, 2004).

Segundo Pfleeger (2004) existem alguns tipos de manutenção:

- **Manutenção corretiva:** realizar modificações para tratar defeitos ou falhas;
- **Manutenção adaptativa:** quando mudanças introduzidas em uma parte do software requerem modificações em outras partes, está ocorrendo a manutenção adaptativa. Ela também pode ser feita para atender a mudanças no ambiente externo ou no hardware;
- **Manutenção perfectiva:** mudanças realizadas para otimizar aspectos do software;
- **Manutenção preventiva:** modificações realizadas no software a fim de prevenir e evitar falhas.

2.3.6 Gerenciamento de Configuração de Software

Esta área de conhecimento é o desenvolvimento e a utilização de padrões e procedimentos para o gerenciamento de sistemas em desenvolvimento. Seus procedimentos definem como processar e registrar mudanças do software, como relacioná-las aos componentes do sistema e os métodos usados para identificar distintas versões dele. Isto é, o gerenciamento de controle possibilita o controle das versões do sistema, assim como o controle do progresso dos artefatos que são criados durante o processo de desenvolvimento (SOMMERVILLE, 2007).

2.3.7 Gerenciamento de Engenharia de Software

Segundo SWEBOK (2004, apud GOMEDE, 2010, acesso em: 05 out. 2013), o

> Gerenciamento de Engenharia pode-se definir como a aplicação das atividades de gerenciamento: planejamento, coordenação, medição, monitoração, controle e documentação, garantindo que o desenvolvimento e a gerência de software sejam sistemáticos, disciplinados e qualificados. O Gerenciamento de Engenharia é tratado sob dois aspectos:
> · Engenharia de Processo: refere-se às atividades empreendidas para geração de políticas, padrões e objetivos organizacionais consistentes;
> · Engenharia de Mensuração: refere-se à atribuição de valores e rótulos às atividades referentes à Engenharia de Software.

O tema central desta pesquisa, o gerenciamento de projetos, é o principal aliado do Gerenciamento da Engenharia de Software, pois através de seus processos e suas áreas de conhecimento descritos no capítulo 4 é possível controlar, monitorar e garantir que as atividades do desenvolvimento sejam qualificadas, organizadas e disciplinadas.

2.3.8 Engenharia de Processo de Software

Esta área de conhecimento se relaciona com as atividades de definição, implementação, gerenciamento, mudanças, melhorias do processo de ciclo de vida de desenvolvimento de software (SWEBOK, apud GOMEDE, acesso em: 06 out. 2013).

2.3.9 Ferramentas e Métodos de Software

Esta área de conhecimento consiste na utilização de ferramentas e métodos que ajudem os integrantes de uma equipe de desenvolvimento no ciclo de vida de software, fazendo com que as atividades do processo de desenvolvimento sejam automatizadas e sistematizadas, o que diminui a ocorrência de erros e maximizam as chances de sucesso do projeto (SWEBOK, apud GOMEDE, acesso em: 06 out. 2013).

2.3.10 Qualidades de Software

Todo software, após ser desenvolvido, possui qualidade quando está em conformidade com os requisitos exigidos pelo cliente. Porém, se os requisitos foram mal especificados o produto pode não ser útil ao usuário.

Da mesma forma que fabricantes buscam modos de garantir a qualidade dos produtos que produzem os engenheiros de software também procuram métodos que garantam que seus produtos são de qualidade e utilidade aceitáveis. Logo, bons desenvolvedores de software devem sempre utilizar uma estratégia para produção de software de qualidade (PFLEEGER, 2004).

De forma bem objetiva, Yourdon (1995) entende que o software tem qualidade quando ele funciona, atende a necessidade do cliente, fica pronto no prazo, merece confiança e ainda pode ser mantido e modificado.

Fisher e Light (1979) se referem a qualidade de software como conjunto de atributos que descrevem o grau de excelência de um sistema computacional.

> A qualidade contempla uma série de objetivos da construção de software, conhecidos como requisitos não-funcionais, tais como extensibilidade, capacidade de manutenção, reutilização do código, desempenho, escalabilidade, usabilidade e confiabilidade nos dados apresentados pela aplicação (ENGHOLM JÚNIOR, 2010, p. 20).

Enfim, podemos dizer que qualidade de software é o resultado de um bom gerenciamento de projetos aliado a uma prática consistente de engenharia de software e possui alguns pontos importantes: **a gestão da qualidade efetiva** (que dá suporte e ajuda a evitar o caos no projeto)**, a utilidade do produto** (fornecendo conteúdo, funções, recursos e confiabilidade de desejo do cliente)**, agregação de valor para fabricante e também ao usuário** (pois um software de alta qualidade deve oferecer benefícios para ambos).

2.4 Considerações sobre o capítulo

Neste capítulo da pesquisa abordamos sobre as necessidades de softwares serem desenvolvidos, assim como foram descritas algumas informações que envolvem as áreas de conhecimento da engenharia de software, destacando que o gerenciamento de projetos tem papel importante em relação a estas áreas.

3 CICLO DE VIDA: MODELOS DE DESENVOLVIMENTO

Neste capítulo serão descritos os modelos de ciclo de vida de desenvolvimento de software, que devem ser empregados de acordo com o porte do sistema a ser criado e o perfil da equipe de trabalho. Além do mais, a escolha de algum ciclo de vida “[...] depende de vários fatores, como: tempo disponível, custo, equipe, dentre outros. Dessa maneira, não existe um modelo ideal, sendo necessário utilizar aquele que mais satisfaça as atuais condições” (ABDO, 2012, acesso em: 15 set. 2013).

Para que os projetos de softwares desenvolvidos atendam as necessidades dos clientes, alguns destes modelos devem ser associados com técnicas de gerenciamento de projetos, visando a organização e otimização dos processos que envolvem o ciclo de vida da engenharia de softwares.

A terminologia “ciclo de vida” pode ser empregada para se referir ao modelo utilizado para trabalhar um projeto de desenvolvimento de software, devendo determinar a ordem dos processos e estabelecer critérios para a transição de uma atividade para outra.

Pfleeger (2004, p.38) nos diz que “todo processo de desenvolvimento de software tem como entrada os requisitos do sistema e como saída um produto fornecido. Muitos modelos foram propostos ao longo dos anos [...]”. Neste capítulo será abordado sobre os modelos precursores e os modelos contemporâneos.

3.1 Modelos Precursores

Os modelos precursores foram aqueles usados como tentativas de organização de trabalho. A seguir abordaremos algumas informações sobre alguns deles.

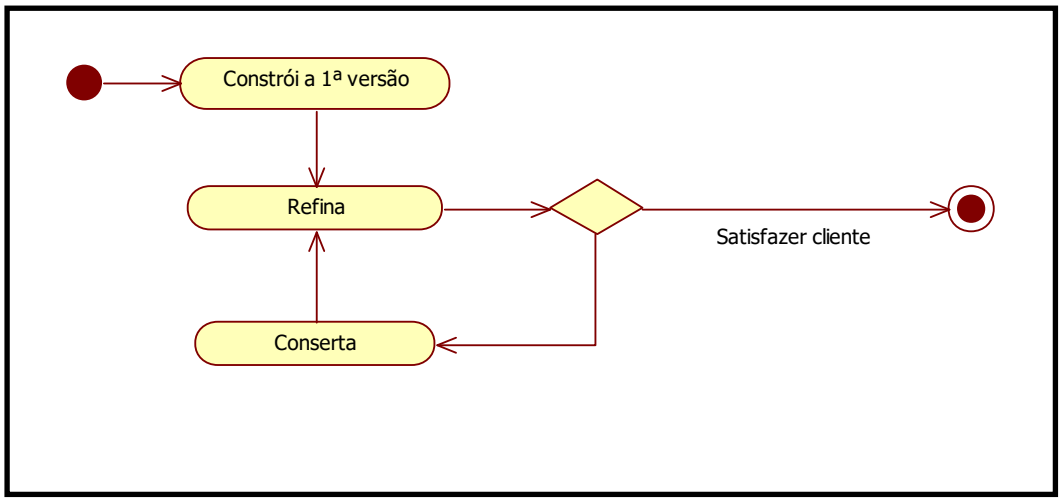
3.1.1 Artesanal

O modelo artesanal é caracterizado por uma dinâmica de trabalho do tipo “tentativa e erro” e foi muito usado antes de surgir a noção de engenharia de software, no fim da década de 60.

Sommerville (2007) afirma que o conceito de engenharia de software foi proposto no ano de 1968, em uma conferência organizada para discutir a chamada ‘crise de software’. Essa crise resultou diretamente da introdução do hardware de computador baseado em circuitos integrados. O software para este hardware sofreu uma expansão em tamanho, distribuição, complexidade e importância, os programas isolados já não supriam mais as necessidades dos usuários, eram necessários sistemas melhores para atender a essas demandas.

Segundo Bräscher e Victorino (2009, acesso em 24 ago. 2013) “nas décadas de 1950 e 1960 os sistemas de software eram bem simples. O desenvolvimento desses sistemas era feito de forma artesanal, pautado em habilidades individuais, sem uma abordagem sistemática específica [...]”, conforme é mostrado na figura 2.

Figura 2 – Modelo Artesanal



Fonte: Adaptado de CDS, acesso em: 13 set. 2013

No modelo artesanal de desenvolvimento de software, havia uma captura informal de requisitos para que o desenvolvedor fizesse a 1ª versão do sistema. Logo após ocorria um refinamento desta versão para, a seguir, haver uma tomada de decisão onde era analisado se este refinamento produziu um resultado satisfatório ao cliente ou não. Geralmente, há a necessidade de consertos e ajustes, fazendo com que este ciclo se repita até o cliente se sentir satisfeito.

No ano de 2013 ainda existem empresas que desenvolvem software de forma artesanal, o que não costuma trazer problemas para desenvolver sistemas de pequeno porte, o qual não exige um esforço muito grande de implementação (CDS, acesso em: 13 set. 2013).

Porém, a utilização deste modelo para desenvolver sistemas de grande porte gerava algumas consequências: dificuldade na produção em escala (sistemas de grande porte), dependência do talento da equipe de desenvolvimento (fazendo haver grandes variações na qualidade do artefato), entre outras.

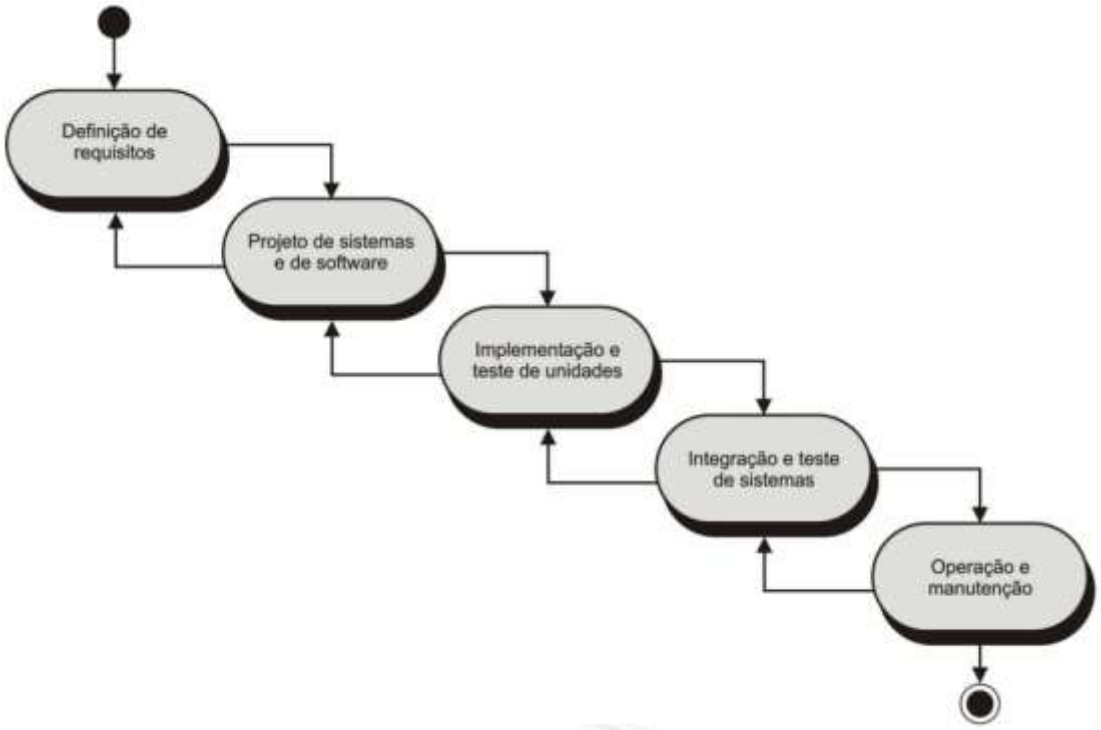
No final da década de 1960, a demanda por sistemas mais complexos mostrou que uma abordagem informal do desenvolvimento de software não gerava os resultados esperados. Novas técnicas e novos métodos foram sistematizados dando origem às metodologias de desenvolvimento de software (BRÄSCHER; VICTORINO, acesso em 24 ago. 2013).

3.1.2 Cascata

O modelo cascata ou clássico foi proposto por Royce em 1970, sendo o único modelo de ciclo de vida com aceitação geral até meados da década de 1980. Ele foi derivado de modelos de atividade de engenharia de software com o objetivo de estabelecer ordem no desenvolvimento de grandes produtos de software, além de ser considerado mais rígido e menos administrativo quando comparado com outros modelos de desenvolvimento (API, acesso em: 14 set. 2013).

Segundo Vargas (2011, acesso em: 14 set. 2013), “neste modelo os processos de desenvolvimento são estruturados de forma que a saída de um processo é a entrada do próximo.” As atividades que envolvem o modelo cascata podem ser observadas na figura 3.

Figura 3 – Modelo Cascata



Fonte: Vargas, 2011

Segundo Sommerville (2007), os estágios que compõe o modelo cascata de ciclo de vida de desenvolvimento de software são assim caracterizados:

- **Definição de requisitos:** são definidos detalhadamente as restrições, os objetivos e funcionalidades do sistema a ser desenvolvido por meio de consulta aos clientes;
- **Projeto de sistemas e software:** os requisitos exigidos pelo cliente são divididos em sistemas de software e hardware. O projeto envolve uma identificação geral da arquitetura e descrições de fatores fundamentais do software a ser desenvolvido;
- **Implementação e teste de unidades:** na implementação o projeto de software é executado em forma de conjuntos de programas ou unidades de programas. Os testes de unidades verificam se elas atendem ao que foi especificado;
- **Integração e teste de sistemas:** neste processo ocorrem a integração e testes dos programas ou unidades de programas como um software completo para verificar se os requisitos foram atendidos. Depois dos testes o software é entregue ao cliente;

- **Operações e manutenção:** é a fase de maior duração no ciclo de vida, pois o sistema é instalado e colocado em operação. Na manutenção é possível corrigir falhas detectadas em fases anteriores e aprimorar a implementação das unidades de sistema, além de ampliar os serviços conforme o surgimento de outros requisitos.

3.1.2.1 Vantagens na utilização do modelo cascata

A utilização do modelo cascata para criar softwares possui algumas vantagens, pois ele torna o processo de desenvolvimento estruturado e bem organizado, tendo uma ordem sequencial nas fases, onde cada uma das fases deve estar finalizada para iniciar outra. Além destas, há facilidade no planejamento, todas as atividades identificadas nos estágios deste modelo são importantes e estão na ordem correta (API, acesso em: 15 set. 2013).

3.1.2.2 Desvantagens na utilização do modelo cascata

Existem também algumas desvantagens neste modelo, pois ele não suporta alterações nos requisitos ao longo do projeto, uma versão do software só ficará pronta no fim do ciclo, atrasos que ocorrerem em qualquer fase afetam todo o processo, entre outras (API, acesso em: 15 set. 2013).

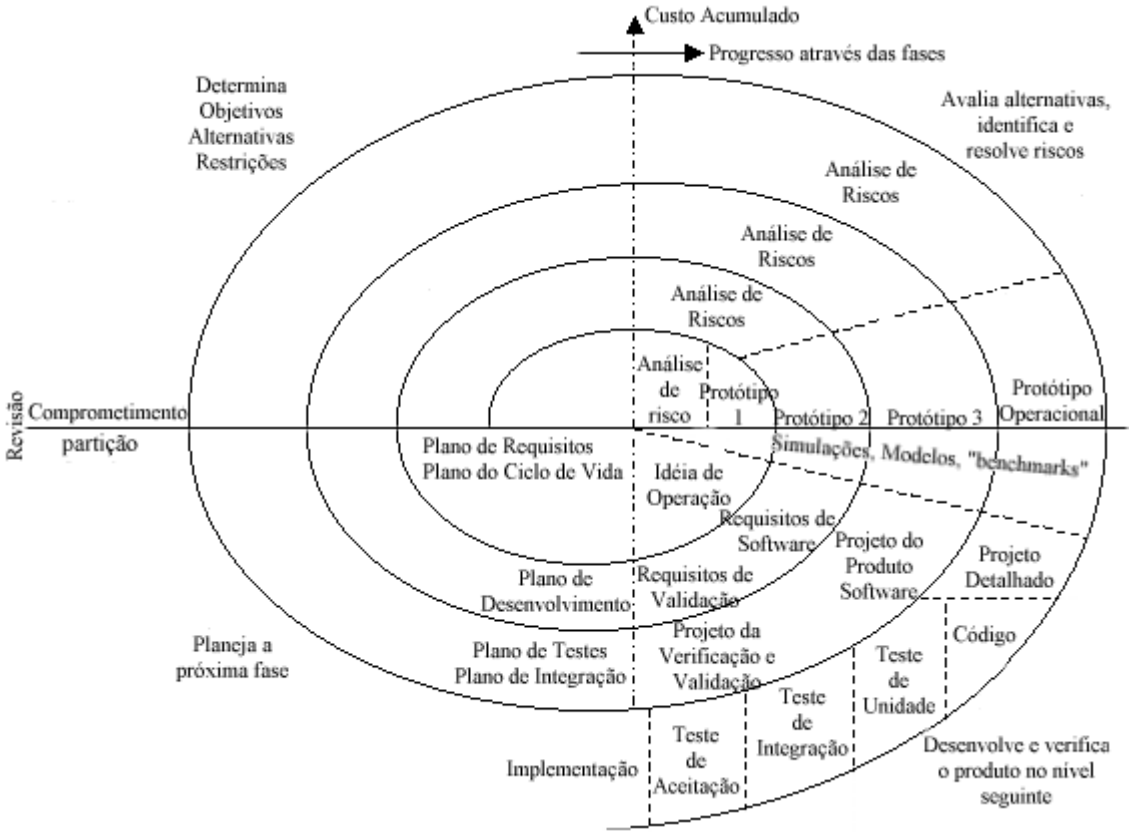
Portanto, possíveis alterações nos requisitos do software a ser desenvolvido podem causar desordem nas demais fases do projeto e, por ter um ciclo sequencial, “o modelo em Cascata é indicado para projetos em que os requisitos são bem estabelecidos para que o processo siga de maneira linear até a última fase de desenvolvimento” (ABDO, 2012, acesso em: 15 set. 2013).

3.1.3 Espiral

Em 1988, Boehm avaliou o processo de desenvolvimento de software a partir do risco envolvido, indicando que um modelo em espiral poderia combinar as atividades de desenvolvimento com o gerenciamento do risco, para que os riscos fossem devidamente minimizados e controlados (PFLEEGER, 2004).

Nesta forma de trabalho proposta ocorre a iteração (repetição) das atividades do modelo cascata, porém de forma organizada, de modo que o software seja construído incrementalmente. Ou seja, o incremento irá estabelecer, em um determinado intervalo de tempo, certa quantia de funcionalidades do sistema. Diferentemente do modelo em cascata, no modelo em espiral o usuário pode explorar as funcionalidades a cada incremento, o que é importante no ponto de vista gerencial, pois é possível perceber problemas nos requisitos não percebidos anteriormente. A figura 4 mostra as tarefas de desenvolvimento do modelo em espiral.

Figura 4 – Modelo em espiral



Fonte: Vargas, 2011

"Cada loop na espiral representa uma fase do processo de software. Dessa forma, o loop mais interno pode estar relacionado à viabilidade do sistema; o próximo loop, à definição de requisitos, o próximo ao projeto de sistema e assim por diante" (SOMMERVILLE, 2007, p. 48).

Ramos (acesso em: 18 set. 2013) nos mostra que cada loop da espiral é composto por quatro setores:

- **Estabelecimento de objetivos:** são estabelecidos objetivos para o projeto, restrições para o processo e o produto, também se projetando um plano de gerenciamento pormenorizado. Além disso, são detectados os riscos do projeto e, dependendo deles, podem ser planejadas variadas estratégias;
- **Avaliação e redução de riscos:** uma análise pormenorizada é realizada para cada um dos riscos detectados. São tomadas algumas atitudes para tentar reduzir os riscos;
- **Desenvolvimento e validação:** logo após a avaliação e redução de riscos é selecionado um modelo de desenvolvimento para o sistema;
- **Planejamento:** é realizada uma revisão no projeto e em seguida há uma tomada de decisão em relação ao prosseguimento do projeto. Se for decidido prosseguir são elaborados planos para o próximo loop.

Segundo Pfleeger (2004, p. 46), no modelo em espiral de desenvolvimento de software, “em cada iteração, a análise de riscos pondera diferentes alternativas em face dos requisitos e das restrições. A prototipação verifica a viabilidade e a adequação, antes que haja a decisão por alguma alternativa”.

O modelo em espiral introduz formalmente a prototipação durante a descoberta de requisitos, ou seja, a equipe desenvolve um protótipo para explorar a lista de requisitos do software.

3.1.3.1 Vantagens na utilização do modelo em espiral

Este modelo possui algumas vantagens em sua utilização, como a associação das melhores características do ciclo de vida em cascata e da prototipação, porém com o foco voltado para o gerenciamento dos riscos e a incorporação de iterações das atividades de forma organizada, usando a prototipação em todos os estágios da evolução do produto como forma de redução dos riscos. Ademais, é uma abordagem mais realística para o desenvolvimento de softwares de grande porte e que capacita o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva (RAMOS, acesso em: 20 set. 2013).

3.1.3.2 Limitações do modelo em espiral

Para a utilização do modelo em espiral para desenvolver softwares é recomendável que a equipe de desenvolvimento seja composta por integrantes experientes, pois desta forma serão minimizadas as dificuldades em controlar os diversos protótipos do software que são feitos. Além disso, uma equipe com integrantes experientes fará uma análise dos riscos relevantes de forma mais apropriada, apesar de aumentar os custos do desenvolvimento para remunerar estes profissionais.

3.1.4 Contribuições dos modelos precursores

Dos modelos precursores de ciclo de vida de desenvolvimento de software, algumas contribuições merecem destaque. No modelo artesanal devemos reconhecer a importância do talento e das habilidades individuais dos profissionais que integram a equipe de desenvolvimento.

No modelo em cascata é relevante a definição da natureza das atividades, ou seja, reconhecer que tarefas como a descoberta de requisitos, análise, design, implementação, testes, entre outras, devem ser efetivamente realizadas. Neste modelo ainda deve ser destacado a importância da disciplina, por ser um ciclo sequencial de atividades.

O modelo em espiral contribui decisivamente como uma forma de trabalho iterativa e incremental, isto é, reconhecemos a necessidade de serem repetidas as atividades e termos incrementos de tempo no qual funcionalidades parciais possam ser exploradas e avaliadas pelo usuário final. Outra contribuição deste modelo é a necessidade da análise e gerenciamento dos riscos.

Segundo Pfleeger (2004, p. 47), “[...] outros modelos de processo podem ser definidos e ajustados de acordo com as necessidades dos usuários, clientes e desenvolvedores”. Ao longo dos anos, diversos padrões de processo de software foram definidos com a finalidade de auxiliar as organizações a criarem softwares de maneira controlada. Por isso, foram surgindo os modelos contemporâneos de desenvolvimento de software, que tendem a preservar de maneira direta ou indireta estas con-

tribuições dos modelos precursores citadas anteriormente. Na próxima seção serão feitos comentários sobre alguns modelos contemporâneos.

3.2 Modelos Contemporâneos

Existem os modelos contemporâneos tradicionais, que possuem como critério predominante de organização dos processos a complexidade dos softwares, e os modelos contemporâneos ágeis, que possuem como critério predominante de organização dos processos a mudança rápida do software ao longo do tempo.

3.2.1 Tradicional

Um exemplo de modelo de processo moderno tradicional é o RUP (Processo Unificado Rational) que se deriva do trabalho sobre o Processo Unificado de Desenvolvimento de Software e da UML (Linguagem Unificada de Modelagem), uma ferramenta que “[...] fornece diagramas de modelagem que podem ser utilizados nos processos do ciclo de desenvolvimento de software” (ENGHOLM JÚNIOR, 2010, p. 68). Este modelo traz uma ênfase nos *workflows* ou fluxos de trabalho, além de definir fases de desenvolvimento que serão abordadas logo a seguir.

O RUP foi criado na década de 90 e usava os conceitos do modelo em espiral como alternativa para solucionar os problemas encontrados no então atual modelo de desenvolvimento de software, o modelo cascata. Ser um produto/processo iterativo, incremental e customizável de Engenharia de Software são algumas das características do RUP, que inicialmente foi desenvolvido e comercializado pela Rational, e a partir de 2003 passou a pertencer à IBM (TAUB JÚNIOR, acesso em: 23 set. 2013).

Sommerville (2007) afirma que o RUP normalmente é descrito a partir de algumas perspectivas:

- **Dinâmica:** exibe as fases do modelo ao longo do tempo;
- **Estática:** exibe as atividades realizadas no processo;
- **Prática:** recomenda boas práticas a serem utilizadas durante o processo.

O RUP possui 4 fases e, segundo Taub Júnior (2009, acesso em: 23 set. 2013) “cada uma [...] compreende um momento distinto dentro do ciclo de vida de um projeto de engenharia de software e, portanto, dão maior ou menor foco em algumas disciplinas, de acordo com a necessidade do projeto no decorrer de sua execução”.

Ainda de acordo com Taub Júnior (acesso em: 23 set. 2013) estas fases são:

- **Início:** nesta fase devem ser estabelecidos alguns objetivos relacionados ao ciclo de vida do projeto, como a visão do projeto: escopo, condições, limites, os critérios de aceitação, entre outros. Também são estimados os potenciais riscos dos requisitos, os custos e prazo total do projeto como um todo e fazer isto detalhadamente na fase de elaboração. Além disso, são identificados as pessoas e sistemas que irão interagir com o software para usar estas informações com forma de avaliar a contribuição do software com o negócio. Para projetos de novos softwares esta fase será mais longa, entretanto, para projetos de software já existentes esta fase será mais breve, mas continuará com a meta de garantir se o projeto é possível e viável;

- **Elaboração:** nesta fase deve ser estabelecido um *framework* de arquitetura para o sistema, instituindo uma base sólida para design e implementação do software. A arquitetura deve considerar os requisitos mais relevantes, podendo ser desenvolvido o plano de projeto e uma avaliação dos riscos principais deste. Para finalizar esta fase, os casos de uso da UML podem ser especificados para que se tenha um modelo de requisitos do software, devendo ainda ser demonstrado que a arquitetura selecionada suportará os requisitos do sistema através de custos e prazos razoáveis;

- **Construção:** nesta fase são estabelecidos os requisitos restantes e se completa o desenvolvimento de software, com base na arquitetura definida na elaboração, além de serem realizados testes de sistema. As partes do software são construídas paralelamente e integradas, criando-se versões utilizáveis a fim de se alcançar uma qualidade adequada para o produto e ao mesmo tempo minimizando os custos de desenvolvimento, evitando retrabalho desnecessário. Ao fim desta fase o software deve estar em funcionamento e com a documentação associada concluída, tornando-se passível de entrega aos usuários;

- **Transição:** a fase final do modelo RUP deve garantir que o software desenvolvido esteja disponível aos seus usuários finais, ou seja, ocorre a transferência do sistema do ambiente de desenvolvimento para o ambiente dos usuários com a implantação deste no ambiente real. Esta fase pode ser dividida em iterações e incluir testes do produto na preparação para seu lançamento, podendo ser feitos alguns ajustes baseados no *feedback* dos usuários e realizar treinamentos com eles. Para finalizar esta fase os objetivos do ciclo de vida do projeto devem ter sido alcançados, devendo haver um sistema de software documentado e funcionando adequadamente no ambiente operacional.

Sommerville (2007) afirma que no modelo RUP cada uma destas fases podem ser realizadas de maneira iterativa, com seus resultados desenvolvidos incrementalmente.

As atividades que ocorrem durante o processo de desenvolvimento são denominadas *workflows* ou fluxo de trabalho. Existem seis *workflows* principais e três *workflows* de apoio que serão descritos a seguir (SENE, acesso em: 24 set. 2013):

- **Modelagem de negócios:** são utilizados os casos de uso de negócios para que sejam modelados os processos de negócios;
- **Requisitos:** são identificados os agentes que interagem com o sistema e são desenvolvidos casos de uso com a utilização da UML para que sejam modelados os requisitos do software;
- **Análise e projeto:** é elaborado e documentado um modelo de projeto com a utilização dos modelos de arquitetura, modelos de componente, modelos de objetos e modelos de sequência;
- **Implementação:** são implementados os componentes de sistema e estruturados em subsistemas. Esse processo é acelerado com a geração automática de código baseado nos modelos de projeto;
- **Teste:** este é um processo iterativo realizado em conjunto com a implementação, logo após a finalização desta;
- **Implantação:** após a elaboração de uma versão do produto, esta é distribuída aos usuários e instalada no local de trabalho;

- **Gerenciamento de configuração e mudança:** é um *workflow* de apoio que gerencia configurações e mudanças do sistema;
- **Gerenciamento de projetos:** também é um *workflow* de apoio e gerencia o desenvolvimento do software;
- **Ambiente:** também é um *workflow* de apoio e está relacionado com a disponibilidade de ferramentas apropriadas de software para a equipe que trabalha com o desenvolvimento.

Segundo Bartoli (acesso em: 26 set. 2013), o RUP é um modelo baseado em um conjunto de princípios e boas práticas de engenharia de software que serão descritos a seguir.

- **Desenvolvimento de software iterativamente:** para se desenvolver um software de grande porte é necessário um tempo relativamente longo, pois não é possível definir o problema do cliente e construir o software em um único passo. O uso de iterações torna-se inevitável, fazendo com que o projeto seja constantemente refinado a cada incremento de tempo e permitindo maior retorno do usuário, auxiliando o desenvolvedor a manter-se focado;

- **Gerenciamento de requisitos:** está relacionado com a identificação e especificação das necessidades do cliente, assim como o acompanhamento das mudanças destes requisitos, analisando o impacto delas no software antes de aceitá-las;

- **Utilização de arquiteturas baseadas em componentes:** um componente está associado com um conjunto de objetos na programação orientada a objetos, fazendo com que o uso de arquiteturas baseadas em componentes para desenvolver softwares possibilite a criação de sistemas facilmente extensíveis, de fácil compreensão e que promova a reusabilidade de software;

- **Modelagem visual de software:** consiste na utilização da UML para modelagem de casos de uso, diagrama de classes e outros objetos. Isto permite que se tenha uma visão geral dos requisitos do software e de possíveis soluções para atender tais requisitos, também proporcionando uma simplificação de um projeto complexo;

- **Verificar a qualidade do software:** O RUP visa auxiliar no controle do planejamento da qualidade, verificando-a na construção de todo o processo e envolvendo todos os membros da equipe de desenvolvimento, a fim de garantir que o software criado atenda aos padrões de qualidade da organização;

- **Controle de alterações no software:** em projetos de software geralmente ocorrem muitas mudanças. O RUP define métodos que controlam e monitoram estas mudanças.

Estas práticas listadas acima são recomendadas para que o andamento de projetos de desenvolvimento de software não seja comprometido.

Portanto,

> o RUP não é um processo adequado a todos os tipos de desenvolvimento, mas representa uma nova geração de processos genéricos. A mais importante inovação é a separação de fases e workflows, e o reconhecimento de que a implantação de software no ambiente do usuário é parte do processo. As fases são dinâmicas e têm objetivos. Os workflows são estáticos e constituem atividades técnicas que não estão associadas a uma única fase, mas podem ser utilizados ao longo do desenvolvimento para atingir os objetivos de cada fase (SOMMERVILLE, 2007, p. 56).

3.2.2 Ágil

A metodologia tradicional de engenharia de software foi utilizada pelas equipes de desenvolvimento por muito tempo, ainda sendo muito usada na atualidade. É uma metodologia que possui rigidez nos seus padrões, ocasionando maior lentidão em relação ao processo de desenvolvimento. Com o passar dos anos as necessidades das pessoas estão mudando, fazendo com que aumente a complexidade dos softwares a serem desenvolvidos para satisfazer tais necessidades. Por causa deste motivo, tornou-se inevitável agilizar o processo de desenvolvimento de software para atender aos clientes que precisam de sistemas desenvolvidos com maior rapidez, o que acabou favorecendo a criação das metodologias ágeis (MODESTO; OLIVEIRA, acesso em: 27 set. 2013).

“Há alguns anos, um grupo de profissionais veteranos na área de software decidiram se reunir em uma estação de esqui, nos EUA, para discutir formas de melhorar o

desempenho de seus projetos" (TELES, 2008, acesso em: 27 set. 2013). Esta reunião composta por 17 especialistas em processos de desenvolvimento de software foi realizada em 2001, entre eles estavam Kent Benck (criador do XP) e Ken Schwaber (criador do Scrum). Nela eles discutiram sobre como melhorar a agilidade no desenvolvimento de softwares e assinaram o Manifesto Ágil (LUCAS, acesso em: 27 set. 2013).

Este Manifesto para o Desenvolvimento Ágil de Software assinado por eles estabelece alguns conceitos e valores, que são abordados por Cruz (acesso em: 27 set. 2013):

- **Indivíduos e interações entre eles mais que processos e ferramentas:** os indivíduos envolvidos com o projeto, sejam eles internos ou externos (equipe de desenvolvimento e clientes, respectivamente), são mais importantes e devem ser mais bem tratados do que máquinas e ferramentas, pois são eles que fazem estas funcionarem;

- **Software em funcionamento mais que documentação abrangente:** metodologias ágeis de software prezam para que haja apenas uma documentação necessária para se ter o software funcionando, pois o produto final é mais relevante do que uma extensa documentação;

- **Colaboração com o cliente mais que negociação de contratos:** é preciso haver um contrato firmado com regras claras e objetivas, além de procurar sempre manter um diálogo transparente com o cliente, para poder contar com sua colaboração;

- **Responder a mudanças mais que seguir um plano:** as mudanças costumam ser a única certeza que existe nos projetos, devendo ser tratadas quando surgirem. Assim sendo, é mais importante responder rápido e corretamente a uma mudança do que seguir um plano inicialmente feito que irá resultar em artefatos sem qualidade.

Sobre estes conceitos, Campos (2011, acesso em: 27 set. 2013) destaca que "[...] mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Isso não quer dizer que devemos eliminar os itens da direita, mas que devemos dar preferência aos itens da esquerda".

O Manifesto Ágil possui 12 princípios que devem ser aplicados e seguidos no gerenciamento ágil de projetos de software:

> - Nossa maior prioridade é satisfazer o cliente, através da entrega adiantada e contínua de software de valor.
> - Aceitar mudanças de requisitos, mesmo no fim do desenvolvimento. Processos ágeis se adequam a mudanças, para que o cliente possa tirar vantagens competitivas.
> - Entregar software funcionando com frequência, na escala de semanas até meses, com preferência aos períodos mais curtos.
> - Pessoas relacionadas à 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 à excelência técnica e bom design, aumenta 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 (LUCAS, 2011, acesso em: 28 set. 2013).

Sobre estes princípios, Campos (acesso em: 28 set. 2013) destaca a importância de se observar que ser Ágil não significa ser radical, nem achar que há apenas uma solução para projetos de desenvolvimento de software. Para ser Ágil o projeto deve conter objetivos bem estabelecidos, a equipe de desenvolvimento deve ser unida, proativa e deve usar soluções simples para problemas simples, garantindo *feedback* contínuo com seus clientes.

Algumas das metodologias ágeis mais populares do mercado e que seguem os princípios e conceitos citados anteriormente são o Scrum e o XP, que serão abordados nas próximas seções.

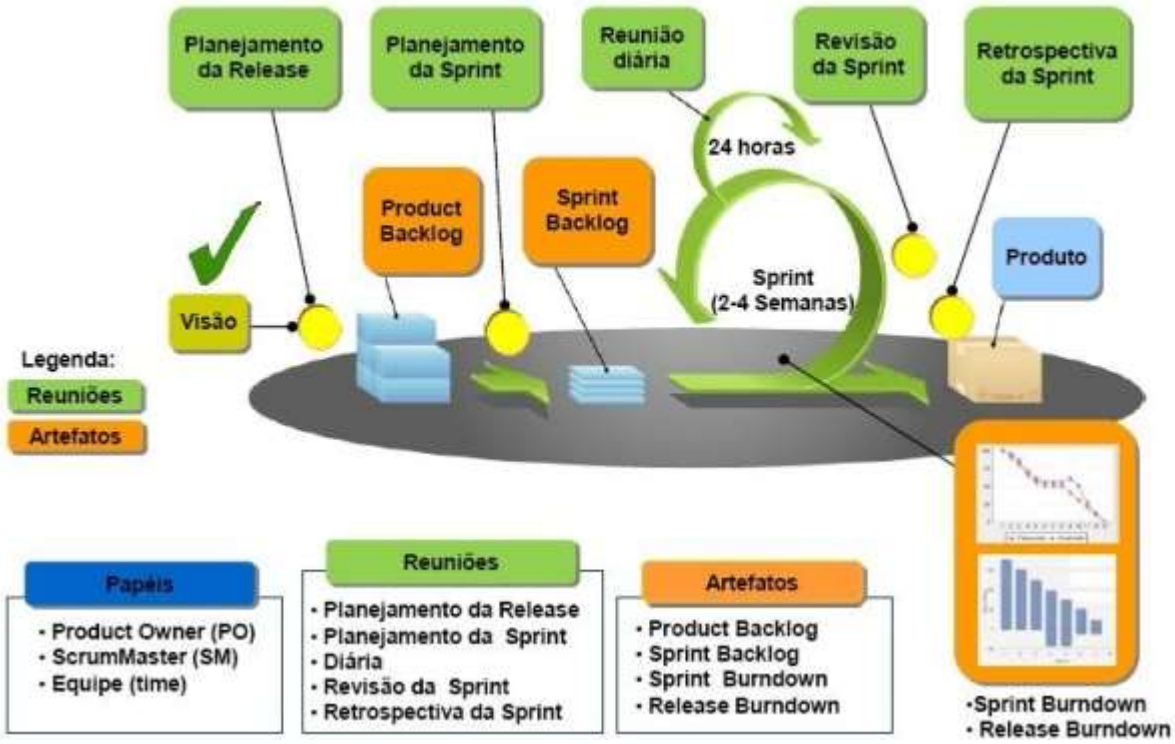
3.2.2.1 Scrum

Esta metodologia foi criada por Ken Schwaber juntamente com Jeff Sutherland. Ken é um dos que assinaram o Manifesto Ágil e fundou uma organização credenciada que oferece aulas, certificações e avaliações sobre o Scrum (DEBONI, acesso em: 29 set. 2013).

Segundo Martins (2012, acesso em: 29 set. 2013) “O Scrum é um framework, criado com base na metodologia ágil, para o gerenciamento e controle de projetos. Ele utiliza um processo iterativo e incremental”, fatores que contribuem para o aperfeiçoamento do controle de riscos e da previsibilidade.

A figura 5 mostra a estrutura do ciclo de vida do Scrum.

Figura 5 – Ciclo de vida do Scrum



Fonte: Martins, 2012

O Scrum é um processo empírico, ou seja, o conhecimento vem da experiência e da tomada de decisões que tem como base o que já é conhecido. Existem três pilares que sustentam o controle desse processo, abordados por Schwaber e Sutherland (acesso em: 30 set. 2013):

- **Transparência:** os aspectos do processo devem ser visíveis para os responsáveis pelos resultados;
- **Inspeção:** os usuários dos Scrum devem inspecionar os artefatos e o progresso do projeto com certa frequência, a fim de identificar variações indesejadas. Porém, a frequência desta inspeção não pode atrapalhar a execução das atividades;
- **Adaptação:** Se na inspeção foi identificado um ou mais processos que desviaram além dos limites aceitáveis, devem ser realizados os devidos ajustes o mais brevemente possível, evitando a ocorrência de outros desvios.

No Scrum, os projetos são divididos em iterações ou ciclos chamados de *Sprints*, que equivalem normalmente de 2 a 4 semanas. Cada *Sprint* representa um tempo determinado em que deve ser executado um conjunto de atividades (MJV, acesso em: 29 set. 2013).

Segundo Schwaber e Sutherland (2011, acesso em: 02 out. 2013) “[...] o Scrum consiste em equipes do Scrum associadas a seus papéis, eventos, artefatos e regras”. A seguir abordaremos sobre cada um destes fatores que compõem o ciclo de vida desta metodologia ágil.

Como podemos observar na figura 6, os artefatos do Scrum incluem o *Backlog* e o *Burndown*. Cruz (acesso em: 01 out. 2013) destaca que o *Backlog* consiste nos requisitos do produto a ser entregue, assim como os meios necessários para atender aos requisitos, desenvolver as funcionalidades e entregar o produto ao cliente. O *Backlog* é considerado dinâmico por estar constantemente mudando para identificar o que o produto necessita para ser útil, apropriado e competitivo. É dividido em duas partes:

- ***Product Backlog*** **(Backlog do Produto):** representa uma lista de requisitos necessários para se desenvolver e lançar um software completo, isto é, representa o

que será entregue após a execução do projeto. Esta lista deve conter itens que devem ser priorizados de acordo com o valor agregado ao produto do cliente;

- ***Sprint Backlog* (Backlog da Sprint):** representa uma lista de requisitos contidos nos objetivos de uma sprint, ou seja, representa tudo que é necessário para desenvolver ou entregar uma parte do software.

Outro artefato da metodologia Scrum é o *Burndown*, um gráfico que representa de forma visual a soma das estimativas dos esforços restantes do *Backlog*, que também possibilita uma comparação com os atuais trabalhos realizados, medindo a velocidade da equipe de desenvolvimento. Possui duas formas de visualização. Uma delas é o ***Release Burndown***, que representa a soma dos esforços restantes do *Backlog* do Produto ao longo do tempo, sendo que o esforço estimado pode estar em qualquer unidade de medida de escolha da equipe, mas normalmente as *sprints* são utilizadas como unidade de medida. A outra forma de visualização é ***Sprint Burndown*,** que representa a quantidade restante de trabalho do *Backlog* da Sprint, ao longo dos dias de duração da sprint. O esforço que for estimado pode estar em qualquer unidade de medida de preferência da equipe, mas normalmente as horas são usadas para este fim (CRUZ, acesso em: 01 out. 2013).

Equipes Scrum são compostas por três papéis (MARTINS, acesso em: 03 out. 2013):

- ***Product Owner* (Dono do Produto):** é o representante do cliente e responsável pelo aspecto funcional do produto**,** isto é, ele deve fazer o gerenciamento do *Backlog* do Produto e também deve garantir que a equipe de desenvolvimento agregue valor ao negócio. A prioridade e a definição das funcionalidades do produto são realizadas pelo *Product Owner* e são descritas em forma de estórias no *Backlog* do Produto. As estórias são descrições claras, objetivas e resumidas de alguma funcionalidade do produto;

- ***Scrum Master* (Facilitador):** deve procurar maneiras de facilitar o trabalho da Equipe Scrum, garantindo que esta siga o fluxo do Scrum adequadamente, garantindo também eficiência e produtividade no trabalho de desenvolvimento. Outras

funções do *Scrum Master* são proteger a Equipe Scrum de interferências externas e remover possíveis impedimentos de realizar o projeto;

- ***Team* (Equipe de Desenvolvimento):** devem ser integradas por profissionais competentes e responsáveis pelo desenvolvimento do produto. É recomendável compor a equipe entre quatro e nove integrantes, fazendo com que ela seja pequena o suficiente para ser ágil e grande o suficiente para possuir um bom rendimento do trabalho.

Quando se utiliza a metodologia ágil Scrum, o ambiente de desenvolvimento exige muita comunicação entre as partes interessadas do projeto, o que é possível com a realização de reuniões ou eventos *time-boxed,* onde todo evento tem uma duração máxima estabelecida. A seguir abordaremos sobre algumas destas principais reuniões e de que forma elas auxiliam a tornar os projetos de desenvolvimento de software mais claros e objetivos (JAMIL, acesso em: 03 out. 2013):

- ***Daily Scrum* (Reunião diária):** acontece diariamente, nela todos que estão envolvidos com o projeto ficam informados sobre o andamento dele. Tem uma duração média de quinze minutos e começa sempre no mesmo horário e local, onde a equipe verifica se foi realizado o planejado da última reunião e faz um planejamento das atividades que serão desenvolvidas nas próximas 24 horas;

- ***Sprint Planning Meeting* (Planejamento da Sprint):** nessa reunião são selecionadas as atividades que serão realizadas durante a *sprint* e se elas podem ser desenvolvidas dentro do tempo da *sprint*. Normalmente tem duração de oito horas para um *sprint* de um mês, podendo variar proporcionalmente ao tempo de duração da *sprint*;

- ***Sprint Review* (Revisão da Sprint):** nessa reunião é revisado o trabalho que foi desenvolvido e, caso seja completada com sucesso uma demonstração é agendada com os *stakeholders* ou as partes interessadas, caso contrário não ocorre apresentação. Normalmente tem duração de quatro horas para um *sprint* de um mês, podendo variar proporcionalmente ao tempo de duração da *sprint*;

- ***Sprint Retrospective* (Retrospectiva da Sprint):** nessa reunião são revistos e reavaliados todos os processos de desenvolvimento usados durante a *sprint*. Além disso, devem ser identificadas possíveis melhorias a serem implementadas na próxima *sprint*. Normalmente tem duração de três horas para um *sprint* de um mês, podendo variar proporcionalmente ao tempo de duração da *sprint*.

Enfim, os artefatos, papéis e reuniões descritos nessa seção compõem o ciclo de vida do Scrum seguindo os três pilares que controlam o processo desta metodologia ágil: transparência, inspeção e adaptação.

3.2.2.2 XP

O XP (*Extreme Programming*) é uma metodologia ágil de desenvolvimento de software, criada nos Estados Unidos por Kent Benck no final da década de 90. Vem fazendo sucesso em diversos países, por auxiliar na criação de softwares com melhor qualidade, que são produzidos em menos tempo e de maneira mais econômica que o habitual. Estes objetivos são alcançados através de um pequeno conjunto de valores, princípios e práticas, diferentes da forma tradicional de se desenvolver software (TELES, acesso em: 07 out. 2013).

O XP possui alguns valores que definem as atitudes das equipes de desenvolvimento e as principais prioridades do projeto. São estes (KUHN; PAMPLONA, acesso em: 07 out. 2013):

- ***Feedback*:** diferentemente da metodologia tradicional, com a utilização da metodologia ágil XP o cliente está em contato com a equipe o tempo todo, estabelecendo as prioridades das funcionalidades que devem ser desenvolvidas. Isto permite que o cliente dê o retorno aos desenvolvedores se o software criado atende suas reais necessidades, ocorrendo o *feedback*. A equipe de desenvolvimento também pode dar *feedback* ao cliente apontando riscos e estimativas, mostrando alternativas de design, entre outros;

- **Comunicação:** para haver sucesso no *feedback* entre cliente e desenvolvedores é necessário uma boa comunicação entre eles, sendo que esta deve ser de

forma direta, clara e objetiva, a fim de evitar mal entendido e que possíveis dúvidas possam ser esclarecidas prontamente;

- **Simplicidade:** os desenvolvedores devem implementar as funcionalidades do software da maneira mais simples possível, facilitando o aprendizado dos clientes na utilização dele, também agilizando o *feedback* e a comunicação;

- **Coragem:** por ser uma metodologia com várias características contrárias ao modelo tradicional, o XP exige que as equipes de desenvolvimento tenham coragem para seguir e colocar em prática os valores, princípios e práticas desta metodologia ágil.

Além destes valores citados, existem um conjunto de práticas que devem ser seguidas pelas equipes que utilizam o XP. Sommerville (2007) descreve características de algumas destas práticas que serão descritas a seguir.

- **Planejamento incremental:** os requisitos são registrados em cartões de estórias, que contém descrições resumidas, claras e objetivas de alguma funcionalidade que o software deve possuir. Estas estórias são incluídas em um release de acordo com sua prioridade relativa e o tempo disponível para implementá-la. Um *release* representa um conjunto de funcionalidades bem definidas que possuem algum valor para o cliente. Os desenvolvedores dividem essas estórias em tarefas, e ao final do ciclo de um *release* é acrescentado um incremento ao software que está sendo desenvolvido;

- **Pequenos *releases*:** através desta prática o XP agrega o máximo de valor econômico ao cliente em um pequeno espaço de tempo. Um projeto que utiliza esta metodologia ágil pode conter um ou mais *releases*, sendo que cada release adiciona funcionalidades incrementalmente ao primeiro;

- **Projeto simples:** o projeto deve ser realizado de maneira mais simples possível, suficiente apenas para atender aos requisitos atuais e nada mais;

- **Desenvolvimento *test-first:*** um *framework* automatizado de teste de unidade é usado para escrever os testes para uma nova funcionalidade antes que esta seja implementada. Os testes de unidade verificam se a implementação está gerando as funcionalidades de forma correta. Existem ainda os testes de aceitação, que verificam se a implementação da funcionalidade está atendendo a necessidade do cliente;

- ***Refactoring:*** esta prática consiste na alteração do código de um software pelos desenvolvedores, para que ele seja melhorado, mas sem que ocorra alteração do comportamento externo do sistema. Isto tornará o código mais simples e fácil de manter;

- **Programação em pares:** os desenvolvedores trabalham em pares, um verificando o trabalho do outro fornecendo apoio para se realizar um bom trabalho. A produtividade de uma equipe que utiliza a programação em pares é relativamente a mesma que uma equipe em que o desenvolvedor trabalha sozinho, porém a qualidade no código fará que se torne mais fácil de mantê-lo, além de gerar outros ganhos como a minimização de falhas que possam passar despercebidas quando se programa sozinho;

- **Propriedade coletiva:** os pares de desenvolvedores trabalham em todas as áreas do sistema, não havendo um responsável para cada parte do software, pois todos os desenvolvedores tem posse de todo o código. Isto faz com que a equipe não se torne dependente de apenas um programador para alterar determinada parte do código quando houver necessidade. Qualquer um da equipe pode alterar qualquer coisa;

- **Integração contínua:** quando o trabalho em uma tarefa for finalizado, deve ser integrado ao sistema como um todo. Isto deve ser feito várias vezes ao dia para que toda a equipe tenha conhecimento do que foi recém-desenvolvido, facilitando o *feedback;*

- **Ritmo sustentável:** grandes quantidades de horas trabalhadas pelos integrantes da equipe de desenvolvimento não são aceitáveis no XP, pois no médio e longo prazo irão reduzir a produtividade;

- **Cliente *on-site:*** um representante do cliente deve estar disponível e presente em tempo integral a fim de apoiar a equipe. No XP, o cliente não é apenas parte interessada, mas também é um membro da equipe de desenvolvimento, responsável por trazer os requisitos do software à equipe de implementação.

Em uma equipe de desenvolvimento de software que utiliza o XP existem alguns papéis desempenhados por seus integrantes (KUHN; PAMPLONA, acesso em: 07 out. 2013):

- **Desenvolvedor:** responsável por analisar, projetar e codificar o software;

- **Gerente de projeto:** responsável pelos assuntos administrativos do projeto, ele deve fazer com a equipe siga técnicas de gerenciamento de projetos a fim de ter organização no planejamento e execução das atividades do ciclo de desenvolvimento;

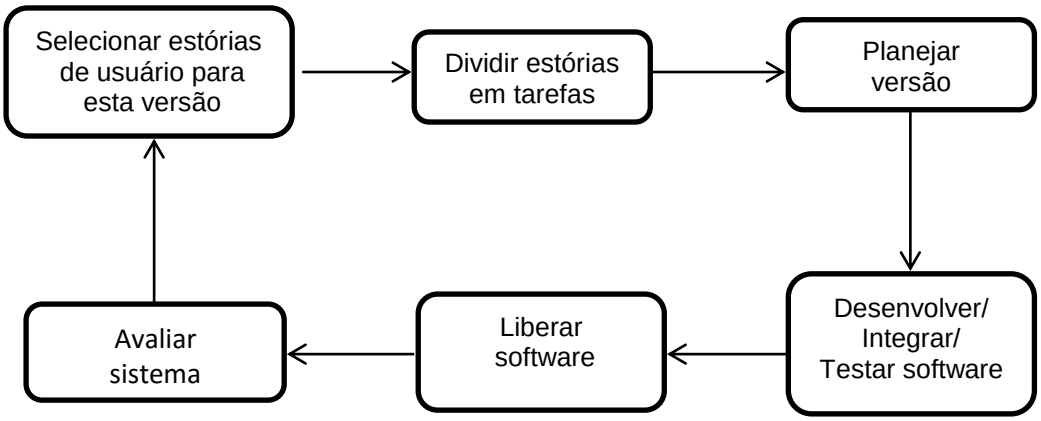
- **Coach:** responsável pelas questões técnicas do projeto, ele deve verificar o comportamento da equipe frente ao processo do XP;

- **Analista de teste:** responsável por garantir a qualidade do software através dos testes escritos;

- **Redator técnico:** responsável por documentar o software, evitando que os desenvolvedores façam este trabalho e permitindo que se dediquem ao trabalho de implementação do sistema.

Na figura 6 pode ser observado o ciclo de um release do XP.

Figura 6 – Ciclo de um release no XP



Fonte: Adaptado de Sommerville, 2007

Enfim, a figura 6 nos mostra que no XP todos os requisitos são expressos em estórias e são implementados através de tarefas. Os programadores trabalham em pares e desenvolvem testes para cada uma das tarefas antes da implementação. Todos estes testes devem ser executados corretamente quando o novo código for integrado ao software, finalizando o release.

Através da descrição das principais características do Scrum e do XP nesta seção da pesquisa, podemos observar que estas duas metodologias ágeis estão alinhadas, porém possuem pequenas diferenças importantes que devem ser consideradas. Equipes Scrum trabalham com iterações (*sprints*) que vão de duas semanas a um mês, não são sujeitos a alterações nos seus *sprints* uma vez definido na reunião de planejamento, e a prioridade das funcionalidades do software a ser desenvolvido é estabelecida por um dos integrantes da equipe de desenvolvimento. Equipes XP trabalham com iterações que vão de uma a duas semanas, são mais flexíveis a mudanças que ocorrerem nas iterações, e a prioridade das funcionalidades do software a ser desenvolvido é estabelecida pelo cliente (LANUSSE, 2010).

Kniberg (2007, apud COSTA, acesso em: 10 out. 2013) destaca que o Scrum fornece bases para a gestão do processo de desenvolvimento, porém não se preocupa com a maneira que será realizado o desenvolvimento. Já o XP recomenda práticas a serem seguidas para o desenvolvimento de um projeto e não se preocupa tanto com a gestão. Sendo assim muitas equipes ágeis utilizam XP e Scrum juntos, de forma que um complete o outro.

3.3 Considerações sobre o capítulo

Neste capítulo foram descritas algumas das principais metodologias de desenvolvimento de software, que foram surgindo nas últimas décadas. Henrajani (2007, apud COSTA, acesso em: 10 out. 2013) afirma que novas metodologias surgem o tempo todo, tornando difícil a escolha de uma corretamente. Isto faz com que a equipe de desenvolvimento tenha que encontrar um equilíbrio adequado para escolher alguma metodologia, considerando as necessidades do cliente, o tamanho do projeto e as metas a serem alcançadas.

4 GERENCIAMENTO DE PROJETOS DE SOFTWARE

Neste capítulo serão abordados os processos e as áreas de conhecimento de gerenciamento de projetos, que podem ser utilizadas em conjunto com alguma das metodologias específicas da engenharia de software descritas neste capítulo, a fim de auxiliar as equipes no correto planejamento e execução das atividades que envolvem o desenvolvimento de sistemas.

.

A principal finalidade da gerência de projetos é melhorar o desempenho do projeto como um todo, pois na grande maioria das vezes é necessário haver mudanças durante o ciclo de vida da engenharia de software: o problema a ser resolvido com a implantação do sistema pode mudar, o cliente pode ter outra ideia, podem surgir novas tecnologias de desenvolvimento, pode haver mudanças no mercado, entre outros. Estes fatores fazem equipes que desenvolvem software utilizar alguma metodologia para auxiliar a gerência de seus projetos.

4.1 O que é Projeto?

Segundo Gervazoni, o projeto é (20--, acesso em: 04 set. 2013), “[...] um processo único, consistindo de um grupo de atividades coordenadas e controladas com data para início e término, que é a chave para se determinar se realmente estamos em um projeto”.

A palavra projeto também pode ser definida como todo resultado obtido ao se “planejar” o que é necessário para o desenvolvimento de um conjunto de atividades: quais são as metas a serem alcançadas, os meios que serão usados para atingi-las, os recursos necessários, onde serão adquiridos e a forma que os resultados serão avaliados. Estes e outros motivos tornam os projetos fundamentais para o sucesso das empresas ou profissionais que trabalham com desenvolvimento de softwares.

4.2 Definindo Gerência de Projetos de Software

Nos dias atuais, as organizações procuram desenvolver projetos para qualquer trabalho que precisa ser feito. Este fato se justifica por causa do dinamismo da econo-

mia, dos recursos finitos disponíveis, tornando necessário haver uma boa gerência destes projetos para que sejam alcançados resultados positivos.

Segundo Pfleeger (2004, p. 63) “um software somente é útil se realizar uma função desejada ou fornecer um serviço necessário. Assim, um projeto começa geralmente quando o cliente o procura para discutir uma necessidade percebida [...]”. Ou seja, o desenvolvedor precisa entender o problema e as necessidades do cliente para então projetar um sistema que resolverá tal problema, sempre levando em conta o tempo e o custo que será empregado para desenvolver o software. Isto requer um cronograma de projeto bem planejado, o que é possível apenas com uma adequada gerência do projeto.

De acordo com Campos e Lima (2009, acesso em: 03 set. 2013),

> [...] o planejamento e execução de projetos nas organizações enfrenta o desafio de desenvolver suas atividades observando os critérios de produtividade, qualidade e cumprimento dos seus planejamentos estratégicos. O gerenciamento de projetos é uma disciplina que tem como objetivo a melhoria do desempenho de um projeto como um todo [...].

Sobre este assunto, Sommerville (2007, p. 61) relata que o

> [...] gerenciamento de projetos de software é uma parte essencial da engenharia de software. Um bom gerenciamento não pode garantir o sucesso de um projeto. No entanto, um mau gerenciamento geralmente resulta em falha do projeto: o software é entregue com atraso, custa mais do que foi originalmente estimado e falha ao atender os requisitos.

Campos e Lima (2009) ainda abordam que o desenvolvimento de uma metodologia de gerência de projetos é de grande importância para qualquer tipo de organização. À medida que o crescimento de projetos gerenciados aumenta a cada dia, um gerenciamento de projetos eficiente consiste em um fator competitivo, que diferencia essas organizações no atual cenário globalizado.

Através da visão de vários estudiosos abordados anteriormente é possível perceber que o sucesso ou a falha de projetos são determinados, na maioria das vezes, pelo gerenciamento de projetos. Isto torna importante haver uma integração da engenharia de softwares com metodologias de gerenciamento de projetos, o que torna mais

fácil as estimativas de prazo, custos e as avaliações de produtividade relacionadas com o desenvolvimento de softwares.

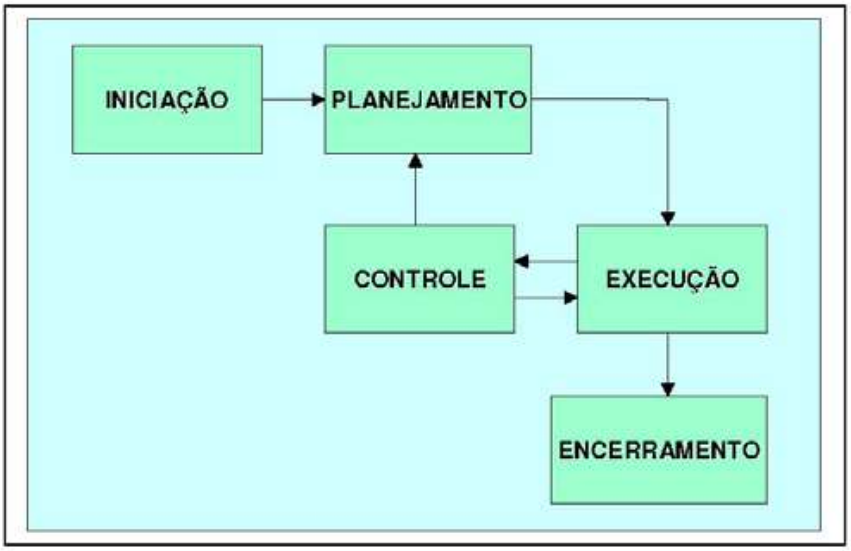
4.3 Guia PMBoK

O gerenciamento de projetos é um conceito importante no que se refere a planejamento de quaisquer atividades de um projeto. Exatamente por isto existe a PMI (Instituto de Gerenciamento de Projetos), a maior associação do mundo sem fins lucrativos para gerenciamento de projetos, e que tem como principal objetivo desenvolver e divulgar métodos de desenvolvimento de projetos. Esta organização realiza uma série de trabalhos e congressos sobre o assunto ao redor do planeta, foi fundada em 1969 e possui, atualmente, milhares de associados que não param de crescer em mais de 170 países. Existem outras instituições que também têm padrões para gestão de projetos, como a ISO, porém apenas o PMI é uma instituição que gera o padrão ANSI (Instituto Nacional Americano de Padrões), uma organização que possui como principal função estabelecer quais normas desenvolvidas devem virar padrão (SOARES, 2008).

Um guia de gerenciamento de projetos desenvolvido pelo PMI e internacionalmente reconhecido por fornecer os conceitos fundamentais nesta área, é o guia PMBoK, disponível gratuitamente para membros do PMI em diversos idiomas, além de ser comercializado nas livrarias mais importantes do planeta (ANZOLIN, 2011).

O conhecimento de gerenciamento de projetos descrito neste guia consiste na descrição das nove áreas de conhecimento que serão abordadas na próxima seção deste capítulo e na definição dos processos de um ciclo de vida de um projeto, como mostrado na figura 7.

Figura 7 – Processos de gerenciamento de projetos



Fonte: Sotille, 2004

4.3.1 Grupos de processos de um projeto segundo o guia PMBoK

Como pode ser observado na figura 7,

> [...] Grupos de Processos é simplesmente um agrupamento dos processos contidos no PMBoK. Existem alguns processos que tem mais a ver com Processos de Iniciação, outros com Planejamento e assim sucessivamente. Os 42 processos do guia PMBoK são divididos nesses 5 grupos de processo. No Processo de Iniciação tem-se dois processos, no processo de Planejamento temos 20 processos, no processo de execução temos 8 processos, no controle temos 10 processos e no encerramento temos 2 processos (MEDEIROS, 20--, acesso em: 08 set. 2013).

4.3.1.1 Iniciação

A iniciação possui dois processos onde são estabelecidos o escopo e os recursos financeiros iniciais, também auxiliando na decisão se o projeto deve ser continuado, adiado ou interrompido.

4.3.1.2 Planejamento

O planejamento possui vinte processos que definem como a execução do projeto deverá ocorrer, onde também é estabelecido seu escopo, o tempo e os gastos que serão utilizados, além de outros planos que sejam necessários.

4.3.1.3 Execução

A execução possui oito processos que servem para executar o que foi estabelecido no planejamento e onde a maior parte dos recursos financeiros devem ser utilizados, podendo ainda ser necessário atualizar ou mudar alguns planos de gerenciamento do projeto.

4.3.1.4 Controle e monitoramento

No controle e monitoramento existem dez processos que servem para acompanhar o desempenho e o progresso do projeto, onde também são identificadas as possíveis mudanças necessárias, assim como as variações em relação ao que foi planejado.

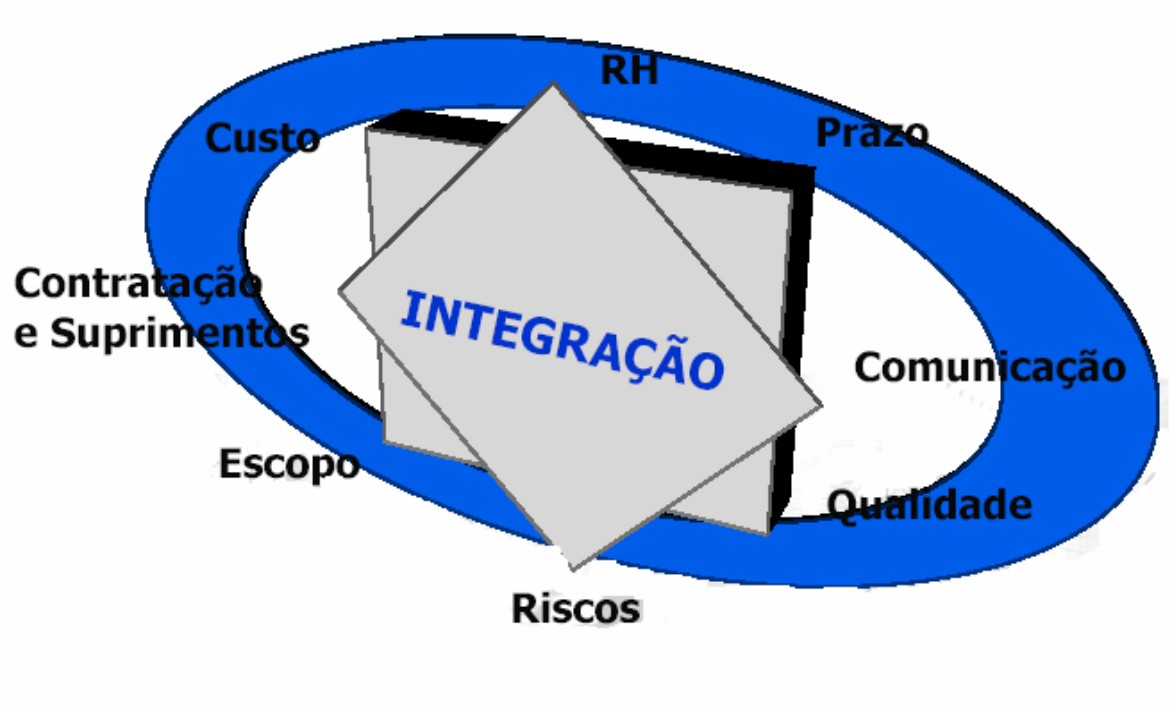
4.3.1.5 Enceramento

O encerramento possui dois processos utilizados para concluir todas as atividades de todos os processos com o objetivo de encerrar o projeto, e onde é documentado o que foi aprendido durante a execução do projeto, ocorre a revisão pós-projeto, devendo haver a aceitação do cliente caso o projeto atenda suas necessidades.

4.3.2 Áreas de conhecimento do Gerenciamento de projetos

Conforme consta no guia PMBoK, o gerenciamento de projeto possui nove áreas de conhecimento técnico que devem ser percorridas para o bom andamento de projetos e que podem ser utilizados nos projetos de desenvolvimento de softwares, como é mostrado na figura 8.

Figura 8 – Áreas do conhecimento do gerenciamento de projetos



Fonte: Sotille, 2004

4.3.2.1 Integração

"[...] O gerenciamento da integração do projeto inclui os processos e as atividades necessárias para identificar, definir, combinar, unificar e coordenar os vários processos e atividades dos grupos de processos de gerenciamento" (ESCRITÓRIO DE PROJETOS, [20--?], acesso em: 09 set. 2013).

A principal meta da integração é realizar as negociações dos conflitos entre alternativas do projeto com a finalidade de atingir ou extrapolar as expectativas e necessidades de todas as partes interessadas, definindo quando e onde concentrar esforço e recursos, resolvendo possíveis problemas que venham surgir durante a execução do projeto (SOTILLE, acesso em: 09 set. 2013).

4.3.2.2 Escopo

Para Sotille (2004, acesso em: 12 set. 2013), no gerenciamento de escopo é possível "[...] definir e controlar o que deve e o que não deve estar incluído no projeto.

Consiste da iniciação, planejamento, definição, verificação e controle de mudanças do escopo.”

A determinação do escopo do software que será desenvolvido deve ser a primeira tarefa de gerência de projeto do software, sendo que “o escopo do produto é composto pela especificação de um conjunto de funcionalidades [...] associada a outras características desejadas [...], tais como desempenho, confiabilidade [...]” (MOTA, [20--?], acesso em: 09 set. 2013).

Além disso, “um dos grandes segredos do gerenciamento de projetos é proteger o seu escopo. Projetos que ficam mudando o escopo durante sua execução têm sérias dificuldades em cumprir o cronograma e estouram o orçamento. [...]” (BARBI, [20--], acesso em: 04 set. 2013).

4.3.2.3 Tempo

O tempo é uma área extremamente importante na gerência de projetos, pois o prazo para finalizar as atividades de execução do projeto é determinante para o sucesso do mesmo.

Mota (acesso em: 09 set. 2013) afirma que depois de estiverem feitas as estimativas de esforço e realizadas em paralelo a alocação de recursos, é possível estipular o tempo cronológico (duração) de cada atividade e do projeto como um todo. Se a estimativa de esforço tiver sido realizada para o projeto como um todo, então ela deverá ser distribuída pelas atividades do projeto.

4.3.2.4 Custos

Os processos que envolvem a área de custos do gerenciamento de projetos visam garantir que o projeto será concluído com o orçamento estipulado na fase inicial.

Sommerville (2007) destaca que os custos de desenvolvimento de software são primariamente os custos de esforço envolvido, de forma que o cálculo do esforço é utilizado para estimativas de custo e tempo. Porém, uma estimativa de custo pode ser

feita antes que os cronogramas sejam criados. As estimativas iniciais podem ser usadas para que seja estabelecido um orçamento para o projeto ou estipular o preço do software ao cliente. No ambiente da engenharia de software, o cálculo do custo total de um projeto envolvem os custos de esforço (pagamento dos desenvolvedores), custos de viagens e treinamentos, custos de hardware e software.

4.3.2.5 Qualidade

Segundo Sotille (2004, acesso em: 09 set. 2013), a qualidade do projeto deve “garantir que o projeto vai satisfazer as exigências para as quais foi contratado. Consiste de planejamento, garantia e controle de qualidade”.

4.3.2.6 Recursos humanos

Segundo Sotille (2004, acesso em: 10 set. 2013), os processos relacionados aos recursos humanos devem “garantir o melhor aproveitamento das pessoas envolvidas no projeto. Consiste de planejamento organizacional, alocação de pessoal e desenvolvimento de equipe”.

Os recursos humanos são fundamentais para o êxito do projeto, pois

> nem toda tarefa é desempenhada pela mesma pessoa ou pelo mesmo grupo. A designação de pessoal para tarefas depende do projeto e das habilidades e experiência do pessoal. Há uma grande vantagem em atribuir diferentes responsabilidades para diferentes grupos de pessoas, oferecendo ‘verificações e balanços’ que podem identificar, antecipadamente, problemas no processo de desenvolvimento (PFLEEGER, 2004, p. 74).

4.3.2.7 Comunicação

Os processos que compõem esta área do gerenciamento de projetos visam garantir que as informações do projeto sejam determinadas, coletadas, armazenadas, organizadas e recuperadas de forma adequada, fazendo com que as pessoas envolvidas interajam de maneira a conduzirem o projeto aos resultados esperados.

Para Mota (acesso em: 10 set. 2013), quando for formada a equipe que irá desenvolver um projeto, é importante levar em conta o tamanho dela. Quanto maior for a quantidade de membros de uma equipe, maiores serão os caminhos possíveis de comunicação, pois o número de pessoas que se comunicam com outras pode afetar negativamente na qualidade do produto resultante.

4.3.2.8 Riscos

Esta área abrange os processos que possibilitam o planejamento, a identificação, o controle, análise quantitativa e qualitativa dos possíveis riscos do projeto.

Por estar diretamente relacionada com o êxito ou o fracasso do projeto, a gerência de riscos é uma das áreas mais críticas da gerência de projetos. Além disso, uma boa gerência de riscos diminui a possiblidade do gerente de projetos e o cliente serem surpreendidos com problemas (RIBEIRO, acesso em: 11 set. 2013).

4.3.2.9 Contratação e suprimentos

Sotille (2004, acesso em: 10 set. 2013) destaca que a principal finalidade desta área é “obter bens e serviços externos à organização executora. Consiste do planejamento de aquisição, planejamento de solicitação, solicitação de propostas, seleção de fornecedores, e administração e encerramento de contratos”.

Enfim, nesta seção foi abordado sobre a forma que o guia PMBoK organiza os processos que compõem o projeto e as áreas que devem ser percorridas para o êxito do mesmo. Este guia de conhecimento específico em gerenciamento de projetos possui cinco edições, sendo a última lançada em 2013, além de ser constantemente atualizado pelos profissionais da área.

4.4 Certificação PMP

A PMI, desde o ano de 1984, vem se empenhando para o desenvolvimento e manutenção de um rígido Programa de Certificação Profissional, baseado em um exame com objetivo de ampliar o reconhecimento das conquistas individuais de gerencia-

mento de projetos e o avanço da profissão nesta área. A credencial profissional mais respeitada e reconhecida no ambiente da gerência de projetos é a Certificação PMP, que pode ser obtida caso o profissional satisfaça alguns requisitos de experiência e educação, concorde com o Código de Conduta Profissional e passe no exame (PMISP, acesso em: 12 set. 2013).

Atualmente, muitas organizações vêm requerendo que seus funcionários possuam Certificação PMP e, no ambiente de desenvolvimento de software, isto é importante para ter respeito no mercado, mais confiança do cliente e um diferencial a mais em relação aos concorrentes.

4.5 Gerente de Projetos

No desenvolvimento de softwares, o projeto traz grandes benefícios por promover organização e planejamento a fim de gerar artefatos que satisfaçam a necessidade dos clientes. Para que isto ocorra, o projeto deve ter um andamento correto e satisfatório, fazendo com que um gerente de projetos tenha um papel determinante e fundamental. Ele deve ser o centralizador de tudo que está dentro do projeto, controlando quaisquer funcionários, clientes e fornecedores que venham envolver-se com o projeto.

Em muitos casos pode haver limitação de recursos, fazendo com que os gerentes tenham que encontrar alguma forma de motivar os integrantes da equipe de desenvolvimento, pois os

> [...] gerentes mais bem-sucedidos na construção de produtos de qualidade, dentro do prazo e orçamentos previstos, são aqueles que adaptam as técnicas de gerenciamento de projeto às características específicas dos recursos necessários, do processo escolhido e das pessoas designadas para o projeto (PFLEEGER, 2004, p.98).

Gervazoni (acesso em: 04 set. 2013) destaca que um gerente de projeto ideal deve possuir algumas habilidades gerenciais (liderança, decisão, comunicação, capacidade de influenciar pessoas, negociação, resolução de conflitos etc.), conhecimento gerencial (técnicas de gerenciamento de projetos e liderança de pessoas), conhecimento técnico dos produtos a serem produzidos, e conhecimento da organização

onde o projeto será executado (cultura organizacional, entre outros). Estas qualidades são imprescindíveis para que o gerente de projetos de desenvolvimento de software o conduza de maneira adequada, a fim de gerar artefatos satisfatórios ao cliente.

Enfim, de forma geral podemos destacar que os gerentes de projetos de software são os responsáveis diretos pelo desenvolvimento de planos e cronogramas de um projeto. Além de supervisionarem o trabalho para garantir que esteja sendo feito conforme os padrões exigidos, eles também devem monitorar o progresso para conferir se o desenvolvimento está no prazo e dentro do orçamento estipulado. Portanto, o gerenciamento de projetos é importante e necessário, pois a engenharia de software profissional está sempre sujeita às restrições de orçamento e de cronograma da organização, tornando o trabalho dos gerentes de projetos de softwares fundamentais para que sejam atendidas estas restrições e que seja entregue aos clientes softwares que contribuam e atendam às suas necessidades (SOMMERVILLE, 2007).

5 A IMPORTÂNCIA DO GERENCIAMENTO DE PROJETOS NO DESENVOLVIMENTO DE SOFTWARES EDUCATIVOS

Segundo Valente (acesso em: 19 out. 2013), a relação entre informática e educação consiste em três fatores: o computador, o software educativo, e o professor treinado para a utilização do computador na sala de aula. Esta relação deve ser estabelecida para que o software educacional funcione como uma ferramenta que torne possível a mudança da qualidade do ensino para melhor. Atualmente, as pessoas vivem em um mundo dominado pela informação e por processos que ocorrem de forma muito veloz. Os fatos e alguns processos específicos que a escola ensina rapidamente se tornam ultrapassados. Assim sendo, ao invés de memorizar informações, os estudantes devem ser ensinados a procurar e a usar as informações. Tais mudanças podem ser introduzidas com a presença do computador, que deve proporcionar as condições para os estudantes exercitarem a capacidade de procurar e selecionar informação, solucionar problemas e aprender independentemente. Isto deve servir a base para o desenvolvimento de software educativo.

Leite (acesso em: 20 out. 2013) destaca que a criação de softwares educacionais deve ter como principal finalidade o desenvolvimento das habilidades dos usuários, e também deve estimular o desafio e a curiosidade destes, ajudando a resolver problemas.

Geralmente, além de envolver em seu desenvolvimento uma equipe multidisciplinar, os produtos de softwares educacionais devem refletir os objetivos educacionais sugeridos e o ambiente de aprendizagem esperado, proporcionando aos usuários situações que estimulem o aperfeiçoamento das habilidades desejadas (CAMPOS; CAMPOS; ROCHA, acesso em: 20 out. 2013).

5.1 Tipos de softwares educativos

No ambiente de aprendizagem com aplicações educacionais, as principais modalidades de software são classificadas por Seabra (2001, apud DANTAS; AQUINO, 2007) da seguinte forma:

- **Exercitação** - oferece treinamento de certas habilidades, possibilitando decorar terminologias de áreas específicas do conhecimento, treinar e resolver problemas;

- **Programas tutoriais** - constituídos por conjuntos de informações pedagogicamente organizados, formando uma espécie de livro animado, um vídeo ou um professor eletrônico; tem a capacidade de organizar o conhecimento de uma área específica, e o aluno pode interagir com os textos do objeto escolhido para estudo por meio dos recursos tecnológicos;

- **Aplicativos** - programas voltados para tarefas mais específicas. Podem ser planilhas eletrônicas, editores de texto, de imagens, de vídeo, de som, programas para criação de apresentações, gerenciadores de bancos de dados, entre outros. Embora não tenham sido criados com fins educativos, devem ser aplicados para educação de alunos;

- **Programas de autoria** - permitem, enquanto extensões avançadas das linguagens, que professores e alunos criem seus próprios programas. Tais programas podem facilitar a criação de apresentações multimídias;

- **Jogos e comunicação** - utilizados para lazer ou entretenimento. Agregados a outras tarefas propostas, podem ter aplicação educacional;

- **Simulações** - funcionam como um ponto forte da utilização de softwares na educação e possibilitam a apresentação de fenômenos, experiências e a vivência de situações complexas ou até perigosas. Proporcionam cenários que se assemelham às situações concretas nas diversas áreas do conhecimento. Nesses ambientes, os alunos podem tomar decisões e comprovar as suas consequências.

5.2 Fases do desenvolvimento de softwares educativos

O desenvolvimento de softwares educativos compreende algumas fases, que são abordadas por Leite (acesso em: 20 out. 2013):

- **Definição do ambiente de aprendizagem** - a escolha de uma área de aprendizagem, para que o software educacional a ser criado auxilie o usuário na ampliação do conhecimento desta área;

- **Análise da viabilidade** - em função dos recursos financeiros necessários e do cronograma estabelecido para entrega, deve ser realizada uma análise de custos para viabilizar o desenvolvimento do software educativo;

- **Planejamento da interface** - representa o mecanismo que estabelecerá a comunicação entre software e usuário e, por isso, deve ser simples e de fácil compreensão para quem utilizar o software educacional;

- **Planejamento do documento** - esta fase compreende a pesquisa e organização do material que será usado no desenvolvimento;

- **Implementação** - compreende a criação do software educacional pelo desenvolvedor ou uma equipe de desenvolvimento;

- **Avaliação** - fase em que são estabelecidos os critérios para a avaliação do software desenvolvido;

- **Validação** – fase em que pode haver coleta de dados junto aos usuários por algum tempo e avaliação contínua, a fim de confirmar e validar que o software educacional auxilia na aprendizagem dos usuários.

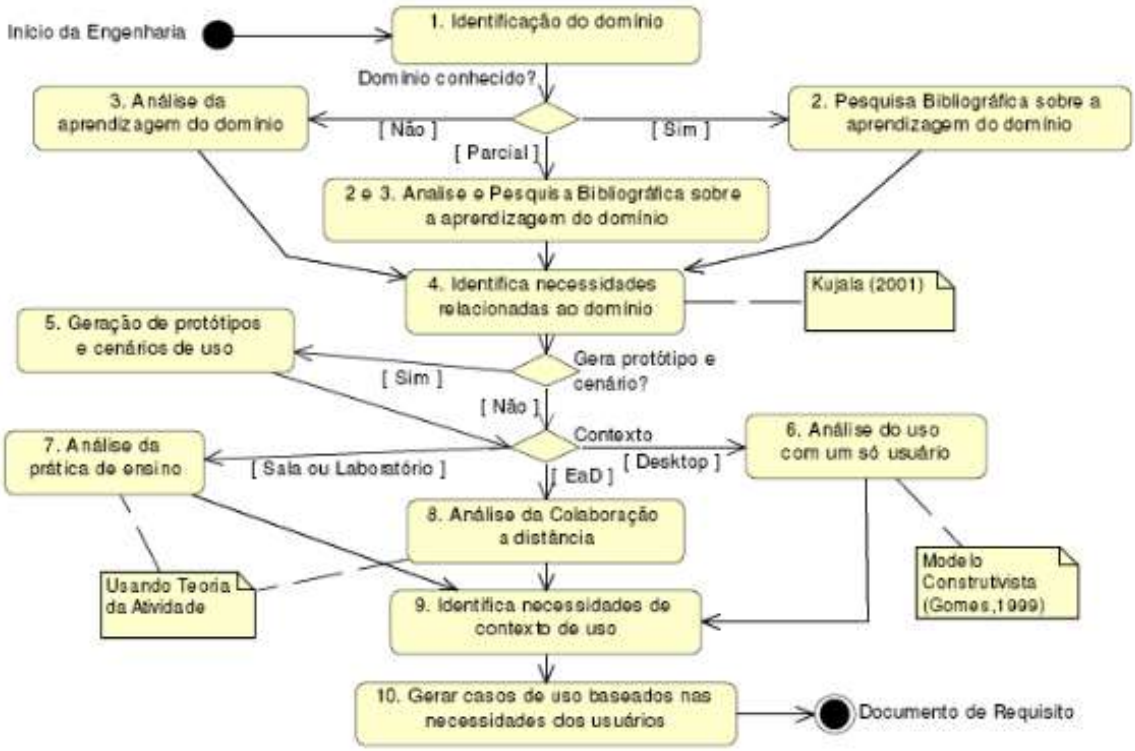
Para haver organização na execução destas fases é importante utilizar os conceitos de gerenciamento de projetos abordados no capítulo 4, associando com alguma das metodologias de desenvolvimento de software descritas no capítulo 3.

5.3 Elicitação de requisitos para softwares educativos

Elicitar requisitos se relaciona com as atividades de captura e obtenção de informações das necessidades que os clientes possuem e que o software a ser desenvolvido deve satisfazer. No caso dos softwares educativos, a elicitação de requisitos se torna ainda mais complexa quando comparada com softwares convencionais, por envolver *stakeholders* (partes interessadas) de distintas áreas de conhecimento (alunos, educadores, empresários, designers, programadores, entre outros). Devido a essa característica dos softwares educativos é recomendada a formação de uma equipe multidisciplinar para desenvolver este tipo de software, pois a presença de especialistas em diversas áreas na equipe aumentará a qualidade no produto final, facilitando e agilizando o processo de desenvolvimento (LACERDA, apud MACIEL, acesso em: 22 nov. 2013).

Além das fases de desenvolvimento citadas na seção 5.2 deste capítulo, o processo para criação de softwares educativos também pode compreender outras atividades. A figura 9 mostra o fluxo de atividades proposto para esta finalidade.

Figura 9 – Diagrama do processo de criação de software educativo



Fonte: Gomes, Vanderley (2003, apud MACIEL, 2008)

A seguir será descrito brevemente algumas informações sobre o fluxo de atividades proposto no modelo de elicitação de requisitos de Gomes e Vanderley (2003, apud MACIEL, acesso em: 23 nov. 2013):

- **Identificação do Domínio:** o domínio deve compreender o campo conceitual a ser apresentado no software educativo (geometria e linguagem de programação, por exemplo);

- **Pesquisa Bibliográfica sobre a aprendizagem do domínio:** se já houver conhecimento sobre o domínio da aprendizagem, os autores recomendam periódicos sobre a aprendizagem de conceitos específicos como a fonte de informações mais acessíveis para pesquisa bibliográfica, embora seja ainda difícil encontrar material acerca da aprendizagem em áreas técnicas mais específicas;

- **Análise da aprendizagem do domínio:** analisando a aprendizagem é possível identificar os requisitos relacionados ao domínio, quando esses não são encontrados na literatura. Além disso, essa análise deve ser realizada através de estudos empíricos, pois as informações desejadas devem estar implícitas na interação dos usuários;

- **Identificar necessidades relacionadas ao domínio:** Com os resultados obtidos na pesquisa bibliográfica e/ou com estudos empíricos associados ao domínio, são identificadas as necessidades dos usuários a serem atendidas pelo software educativo;

- **Geração de protótipos e análise cenários de uso:** a criação de protótipos permite que os usuários testem as interfaces do software antes que este seja implementado, podendo assim validar os requisitos ou refiná-los. Esta tarefa é opcional, podendo ser ou não realizada de acordo com o projeto;

- **Análise do uso do usuário aluno:** Nesta tarefa são utilizadas técnicas para analisar a interação do aluno com a máquina no contexto de uso da aplicação. O resultado desta análise irá orientar o projeto do software;

- **Análise da prática de ensino:** geralmente, softwares educativos são voltados para atividades que devem ser realizadas por apenas uma pessoa (o aluno). Esta atividade aborda um paradigma diferente, onde é destacada a importância de se observar a atividade normal de um professor num laboratório de informática educativa para, a partir dessa observação, originar requisitos que promovam o design de ferramentas adequadas;

- **Análise da colaboração à distância:** voltada para descobrir requisitos de softwares de ensino à distância. A análise da colaboração é feita em estudos sistemáticos relacionados com a observação e também podem ser analisados pela teoria da atividade;

- **Identificar necessidades do contexto de uso:** logo após o estudo do contexto é possível identificar os requisitos ligados a ele. Essas informações são então organizadas de maneira que os designers possam entender e traduzir as necessidades de contexto e de domínio para casos de uso;

- **Gerar casos de uso baseado nas necessidades do usuário:** casos de uso auxiliam designers a ter uma visão coerente do produto. Eles definem atores e detalham os serviços que serão implementados no software educativo, de forma que todos os requisitos sejam atendidos por um ou mais casos de uso.

5.4 Integrantes de uma equipe de desenvolvimento de softwares educativos

Independentemente de qual metodologia será utilizada para desenvolver um software educativo, a equipe para desempenhar tal atividade deve ser composta pelos seguintes integrantes (BENITTI; SEARA; SCHLIDWEIN, 2005):

- **Profissionais da educação:** podem ser representados pelos professores, pois todo software educativo está relacionado a uma realidade de ensino. Tal participação irá validar o conteúdo do software, uma vez que professores são usuários em potencial.

- Outros profissionais da educação que também devem participar de projetos são os pedagogos, especialistas na área de aprendizagem;

- **Profissionais na área computacional:** representados por especialistas em Engenharia de Software e programação, que possuem conhecimentos de técnicas aplicadas ao ciclo de desenvolvimento (técnicas de testes de software, métodos de especificação de requisitos como o abordado na seção 5.3 desta pesquisa, entre outros). Outros especialistas devem ter o domínio de linguagens de programação e tecnologias específicas que podem ser utilizadas pela equipe (interface em 3D, por exemplo);

- **Designer:** profissional que deve possuir o conhecimento na área de usabilidade e dominar ferramentas para desenvolvimento de animações, sons, gráficos. O designer será o responsável pela criação da interface do software educativo, ou seja, deverá criar a comunicação entre sistema e usuário;

- **Usuário (aluno):** representa aquele que utilizará o sistema, possibilitando uma amostragem significativa relacionada ao contexto de aplicação do software. Assim como os professores, os alunos também têm a atuação no processo de desenvolvimento centrada na validação do software educativo.

5.5 A atuação das metodologias de gerenciamento de projetos no processo de desenvolvimento de softwares educativos

Os diversos softwares educacionais criados na atualidade utilizam recursos hipermídia, consentindo o trabalho individual ou coletivo, utilizam técnicas de Inteligência Artificial em distintas escalas e podem ser exploradas pelos professores de várias maneiras, a fim de auxiliar no aprendizado dos alunos. Estas possibilidades fazem com que o projeto destes softwares seja complexo. Na atualidade, é inviável desenvolver um bom software educacional sem a ajuda de uma equipe multidisciplinar (como abordado na seção 5.4 desta pesquisa) composta por especialistas da área de domínio da aplicação, especialistas em Informática na Educação (principalmente na área de gerenciamento de projetos e desenvolvimento desta modalidade de programas) e em Engenharia de Software. Os novos ambientes exigem um conjunto de

funcionalidades necessárias para o design do software que deverá atender os aspectos pedagógicos exigidos. Para isso, deve ser realizado um levantamento criterioso dos requisitos identificados pelos professores especialistas. Entretanto, esta não é uma atividade de fácil execução dado o perfil e conhecimento tão diversificado de equipes de desenvolvimento interdisciplinares. Logo, faz-se necessário que se utilize uma metodologia para guiar e organizar as fases do projeto de desenvolvimento do software educacional (GIRAFFA; MARCZAK; PRIKLADNICKI, 2005).

Existem diversas metodologias que podem ajudar equipes na gerência de projetos de desenvolvimento de software educacional e que foram abordadas no capítulo 3 desta pesquisa. Uma delas é o RUP, uma metodologia que utiliza o UML para modelagem de dados, além de ser baseada em *workflows* e fases de desenvolvimento. Porém existe rigidez em seus padrões, o que pode tornar o processo de desenvolvimento mais lento. Leitão (2010) afirma que as maiores dificuldades enfrentadas pelas empresas que desenvolvem software (como os atrasos na entrega do projeto, produtos sem qualidade e custos elevados) são originados, entre outros fatores, pela ausência ou mau gerenciamento nos processos. Por estes motivos as metodologias ágeis, como o Scrum e o XP tornam-se mais adequadas para serem usadas no desenvolvimento de softwares educativos.

Contudo, as organizações têm grandes problemas na implantação de métodos de desenvolvimento. A maior dificuldade em um processo de implantação é a resistência à mudança. Outra dificuldade é a inevitável baixa na produtividade, em um primeiro momento. Isto acontece pelo fato de que ocorre uma mudança na forma pela qual os profissionais estão habituados a executar suas tarefas. Toda mudança requer um tempo de adaptação que varia de acordo com o perfil da equipe de desenvolvimento. Enquanto a nova metodologia implantada não estiver assimilada, os desenvolvedores encontrarão mais dificuldades em exercer suas atividades da maneira com a qual estavam acostumados. Em meio às abordagens até hoje usadas, os métodos ágeis abordados na seção 3.2.2 desta pesquisa, devem receber destaque, principalmente em relação às mudanças, pois oferecem, em algumas situações, respostas rápidas às novas exigências de desenvolvimento: requisitos que mudam constantemente, não são totalmente esclarecidos e a entrega do produto com valor aceitável (LEITÃO, 2010).

6 CONSIDERAÇÕES FINAIS

No ambiente do desenvolvimento de softwares educativos, alguns dos principais obstáculos enfrentados pelas equipes são os atrasos na entrega do projeto, produtos com qualidade ruim e custos elevados. Tais dificuldades são originadas, em grande parte, pela ausência ou mau gerenciamento dos projetos.

Ao longo da pesquisa foi possível perceber que, com o passar do tempo, a complexidade dos softwares a serem desenvolvidos foi aumentando por causa da constante mudança das necessidades das pessoas e empresas, pois os softwares são criados para resolver problemas e atender necessidades. Os sistemas já chegaram a ser desenvolvidos de maneira artesanal, ou seja, sem nenhuma técnica de gerenciamento de projetos, o que não é possível na criação de softwares de grande porte. O surgimento de modelos como o cascata, o espiral e logo após o RUP, Scrum e XP, nos mostram que a evolução e o surgimento de metodologias para se desenvolver sistemas aliados com técnicas de gerenciamento de projetos tornaram-se necessários para atender a complexidade na implementação de softwares e a necessidade de desenvolvê-los mais rapidamente.

O guia PMBoK, da PMI, nos fornece os principais conceitos na área de gerenciamento de projetos e a sua utilização por equipes de desenvolvimento de sistemas juntamente com alguma metodologia específica de engenharia de software, tornou-se indispensável para haver organização no que diz respeito ao planejamento e execução das atividades que envolvem a criação de softwares educativos. As equipes devem considerar o tamanho do projeto, o tempo para desenvolvimento, as metas a serem alcançadas, as habilidades que serão melhoradas com a utilização dos softwares educativos criados, entre outros, para escolha de alguma das metodologias existentes ou alguma que venha a surgir, procurando utilizar os conceitos de gerenciamento de projetos como auxílio no desenvolvimento.

Portanto, entre todas as metodologias citadas nesta pesquisa, as metodologias ágeis, como o Scrum e o XP, são as que mais se adequam para proporcionar organização no planejamento e execução dos processos que se relacionam com o de-

senvolvimento de softwares educativos. Isto se deve principalmente por oferecerem, em relação às mudanças, respostas rápidas às novas exigências de desenvolvimento: requisitos que mudam constantemente ou que não são totalmente esclarecidos, o software deve ser feito com o menor custo e tempo possível, entre outros.

6.1 Trabalhos futuros

Esta pesquisa oferece como sugestão para trabalhos futuros o estudo dos conceitos de gerenciamento de projetos aplicado no desenvolvimento de outros tipos de sistemas, ou aplicado ao desenvolvimento de softwares em geral.

7 REFERÊNCIAS

ABDO, Samyr. **Ciclo de vida de software – modelo em cascata**. Disponível em: <http://tiemprosa.com.br/ciclos-de-vida-do-software/>. Acesso em: 15 set. 2013.

ANZOLIN, Wanderson. O que é o guia PMBoK?. **Blog Wanderson Anzolin** (Blog). [S.l.]: Wanderson Anzolin. 26 jun. 2011. Disponível em: <http://wandersonanzolin-gp.blogspot.com.br/2011/06/o-que-e-o-guia-pmbok.html>. Acesso em: 07 set. 2013.

API. **Modelo cascata ou clássico**. [S.l.]: [20--?]. Disponível em: <http://www.api.adm.br/GRS/referencias/t1_g13.modeloCascata.pdf>. Acesso em: 14 set. 2013.

BARBI, Fernando C. **Os 7 passos do gerenciamento de projetos.** [S.l.]: 20--. Disponível em: <http://www.microsoft.com/brasil/msdn/tecnologias/carreira/gerencprojetos.mspx>. Acesso em: 04 set. 2013.

BARTOLI, Joel de. **RUP – Rational Unified Process.** [S.l.]: 2013. Disponível em: <http://www.guiafar.com.br/portal/index.php/pt/artigos/tecnologia-da-informacao/108-rup>. Acesso em: 26 set. 2013.

BENITTI, Fabiane Barreto Vavassori; SEARA, Everton Flávio Rufino; SCHLIDWEIN, Luciane Maria. **Processo de desenvolvimento de software educacional:** proposta e experimentação. Porto Alegre: 2005. Disponível em: <http://www.seer.ufrgs.br/renote/article/viewFile/13849/8025>. Acesso em: 25 nov. 2013.

BRÄSCHER, Marisa; VICTORINO, Marcio. **Organização da informação e do conhecimento, engenharia de software e arquitetura orientada a serviços:** uma Abordagem Holística para o Desenvolvimento de Sistemas de Informação Computadorizados. [S.l.]: 2009. Disponível em: <http://www.dgz.org.br/jun09/Art_03.htm>. Acesso em: 24 ago. 2013.

CAMPOS, César. **Manifesto Ágil – e seus princípios.** [S.l.]: 2011. Disponível em: <http://profcesarcampos.com.br/?p=202>. Acesso em: 27 set. 2013.

CAMPOS**,** Fernanda; CAMPOS, Gilda; ROCHA, Ana Regina. **Dez etapas para o desenvolvimento de software educacional do Tipo Hipermídia.** Rio de Janeiro: [20--?]. Disponível em: <http://lsm.dei.uc.pt/ribie/docfiles/txt200352152926dez%20etapas%20para%20o%20desenvolvimento.pdf>. Acesso em: 20 out. 2013.

CAMPOS, Lídio Mauro Lima de; LIMA, Alberto Sampaio. **Gerenciamento de projetos de desenvolvimento de software com o RUP e o PMBOK.** [S.l.]: 2009. Disponível em: <http://www.aedb.br/seget/artigos09/163_seget_2009.pdf>. Acesso em: 03 set. 2013.

CDS, Condomínio de Soluções Corporativas. **Engenharia de Software**. [S.l.]: [20--?]. Disponível em: <http://www.cds.com.br/?p=2669>. Acesso em: 13 set. 2013.

COSTA, Leandro. O sprint e os artefatos do Scrum. **Semeru Blog** (Blog). [S.l.]: Leandro Costa. 04 abr. 2011. Disponível em: <http://www.semeru.com.br/blog/?p=69>. Acesso em: 10 out. 2013.

CRUZ, Fábio. Artefatos Scrum. **Blog Fábio Cruz** (Blog). [S.l.]: Fábio Cruz. 2013. Disponível em: <http://www.fabiocruz.com/outros/artefatos-scrum/>. Acesso em: 01 out. 2013.

CRUZ, Fábio. Manifesto Ágil. **Blog Fábio Cruz** (Blog). [S.l.]: Fábio Cruz. 2013. Disponível em: <http://www.fabiocruz.com/outros/manifesto-agil/>. Acesso em: 27 set. 2013.

DANTAS, Geórgia Geogletti Cordeiro; AQUINO, Mirian de Albuquerque. **Aprendendo com o uso de softwares educativos para ativar inteligências múltiplas (IM).** [S.l.]: 2007. Disponível em: <http://www.seer.ufrgs.br/EmQuestao/article/view/55/1576>. Acesso em: 19 out. 2013.

DEBONI, Eduardo. Guia do Scrum 2011 em português. **Blog Eduardo Deboni** (Blog). [S.l.]: Eduardo Deboni. 02 out. 2011. Disponível em: <http://www.eduardodeboni.com/blog/?p=512>. Acesso em: 29 set. 2013.

ENGHOLM JÚNIOR, Hélio. **Engenharia de software na prática.** São Paulo: Novatec Editora, 2010.

ESCRITÓRIO DE PROJETOS. **Gerenciamento da integração do projeto**. [S.l.]: [20--?]. Disponível em: <http://escritoriodeprojetos.com.br/gerenciamento-da-integracao-do-projeto.aspx>. Acesso em: 09 set. 2013.

GERVAZONI, Thiago Pastorello. **Conceitos Básicos de Gerenciamento de Projetos.** [S.l.]: 20--. Disponível em: <http://www.linhadecodigo.com.br/artigo/1167/conceitos-basicos-de-gerenciamento-de-projetos.aspx>. Acesso em: 04 set. 2013.

GIRAFFA**,** Lucia; MARCZAK, Sabrina; PRIKLADNICKI, Rafael. **PDS-E:** em direção a um processo para desenvolvimento de software educacional. Porto Alegre: 2005. Disponível em: <http://www.br-ie.org/pub/index.php/wie/article/view/867/853>. Acesso em: 20 out. 2013.

GOMEDE, Everton. Áreas de Conhecimento segundo o SWEBOK. **Blog Everton Gomede** (Blog). [S. l.]: Everton Gomede. 13 ago. 2010. Disponível em: <http://evertongomede.blogspot.com.br/2010/08/areas-de-conhecimento-segundo-o-swebok.html>. Acesso em: 28 ago. 2013.

ISD BRASIL. **Perguntas frequentes:** O que é SCAMPI?. [S.l.]: [20--?]. Disponível em: <http://www.isdbrasil.com.br/o-que-e-scampi.php>. Acesso em: 12 ago. 2013.

JAMIL, Vítor. Reuniões da Metodologia Scrum. **Blog Scriptcase** (Blog). [S.l.]: Vítor Jamil. 07 jun. 2013. Disponível em: <http://www.scriptcase.com.br/blog/reunioes-da-metodologia-scrum/>. Acesso em: 03 out. 2013.

KUHN, Giovane Roslindo; PAMPLONA, Vitor Fernando. **Apresentando XP. Encante seus clientes com Extreme Programming.** [S.l.]: 2009. Disponível em: <http://javafree.uol.com.br/artigo/871447/Apresentando-XP-Encante-seus-clientes-com-Extreme-Programming.html>. Acesso em: 07 out. 2013.

LANUSSE, Andreano. Metodologias ágeis – Scrum ou Extreme Programming(XP)?. **Blog Andreano Lanusse** (Blog). [S.l.]: Andreano Lanusse. 02 ago. 2010. Disponível em: <http://www.andreanolanusse.com/pt/metodologias-ageis-scrum-ou-extreme-programming-xp/>. Acesso em: 08 out. 2013.

LEITÃO, Michele de Vasconcelos. **Aplicação de Scrum em Ambiente de Desenvolvimento de Software Educativo**. Recife: 2010. Disponível em: <http://tcc.ecomp.poli.br/20101/TCC_final_Michele.pdf>. Acesso em: 20 out. 2013.

LEITE, Letícia Lopes. **Engenharia de software educacional.** [S.l.]: [20--?]. Disponível em: <http://www.inf.pucrs.br/~lleite/seII/material/aula1.pdf>. Acesso em: 20 out. 2013.

LUCAS, Jônatas. O manifesto Ágil. **Blog JHONYWEBMASTER** (Blog). [S.l.]: Jônatas Lucas. 17 nov. 2011. Disponível em: <http://jhonywebster.blogspot.com.br/2011/11/o-manifesto-agil.html>. Acesso em: 27 set. 2013.

MACIEL, Diogo Rodrigues. **Análise de requisitos para software educativo**. Recife: 2008. Disponível em: <http://www.cin.ufpe.br/~in1020/arquivos/monografias/2008-1/Monografia_ER%20-%20drm.pdf>. Acesso em: 22 nov. 2013.

MARTINS, Luciana. Metodologias ágeis: entenda o Scrum. Disponível em: <http://www.lg.com.br/canais/mais-ti/entrevistas/metodologias-ageis-entenda-o-scrum>. 21 dez. 2012. Entrevista concedida ao Mais TI da LG Sistemas. Acesso em: 29 set. 2013.

MEDEIROS, Higor. **Grupos de processos segundo o PMBoK.** [S.l.]: [20--]. Disponível em: <http://www.devmedia.com.br/grupos-de-processos-segundo-o-pmbok/27106>. Acesso em: 08 set. 2013.

MJV. Scrum? O que é? Como funciona?. **Blog Tecnologia e Inovação** (Blog). [S.l.]: 16 jul. 2009. Disponível em: <http://www.inovatividade.com/metodologias/scrum-o-que-e-como-funciona>. Acesso em: 29 set. 2013.

MODESTO, Jéssica; OLIVEIRA, Caique. **Metodologia Ágil – O que é e como surgiu.** [S.l.]: 2010. Disponível em: <http://nocoesengsw.blogspot.com.br/2010/06/metodologia-agil-o-que-e-e-como-surgiu.html>. Acesso em: 27 set. 2013.

MOTA, Kleber. **Gerência de projetos de software**. [S.l.]: [20--?]. Disponível em: <http://www.fattocs.com.br/livro-apf/citacao/KleberMota.pdf>. Acesso em: 09 set. 2013.

PFLEEGER, Shari Lawrence. **Engenharia de software:** teoria e prática. Tradução Dino Franklin; revisão técnica Ana Regina Cavalcante da Rocha. 2. ed. São Paulo: Prentice Hall, 2004.

PMISP. **Certificação PMP**. [S.l.]: [20--?]. Disponível em: <http://www.pmisp.org.br/certificacao/pmp>. Acesso em: 12 set. 2013.

RAMOS, Ricardo Argenton. **Processo de desenvolvimento de software.** [S.l.]: [19--?]. Disponível em: <http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ESI2009_2/Aula02.pdf>. Acesso em: 10 ago. 2013.

RAMOS, Ricardo Argenton. **Processos de desenvolvimento de software.** [S.l.]: [20--]. Disponível em: <http://www.univasf.edu.br/~ricardo.aramos/disciplinas/ES_I_2010_2/Aula%2003%20(processo%20desenv%20de%20software).pdf>. Acesso em: 18 set. 2013.

SOARES, Ricardo. **O que é PMI?.** [S.l.]: 2008. Disponível em: <http://www.programabrasil.org/o-que-e-o-pmi/#.UiiVosakqOd>. Acesso em: 05 set. 2013.

RIBEIRO, Wankes Leandro. **Entendendo o básico do Gerenciamento dos riscos de um projeto**. Disponível em: <http://www.wankesleandro.com/2009/11/entendendo-o-basico-do-gerenciamento.html>. Acesso em: 11 set. 2013.

SCHWABER, Ken; SUTHERLAND, Jeff. **Um guia definitive para o Scrum:** as regras do jogo**.** [S.l.]: 2011. Disponível em: <https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/Scrum%20Guide%20-%20Portuguese%20BR.pdf>. Acesso em: 30 set. 2013.

SENE, Rafael Peria de. **RUP – Primeiros passos.** [S.l.]: 2011. Disponível em: <http://www.tiespecialistas.com.br/2011/02/rup-primeiros-passos/#.UkGYsYakqOc>. Acesso em: 24 set. 2013.

SOMMERVILLE, Ian. **Engenharia de software.** Tradução: Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edílson de Andrade Barbosa; revisão técnica: Kechi Kirama. 8. ed. São Paulo: Pearson Addison-Wesley, 2007.

SOTILLE, Mauro. **Gerenciamento de projetos na engenharia de software.** [S.l.]: 2004. Disponível em: <http://www.pmtech.com.br/artigos/Gerenciamento_Projetos_Software.pdf>. Acesso em: 03 set. 2013.

SOUZA, Washington. O que é CMMI?. **Blog CMMI** (Blog). [S. l.]: Washington Souza. 22 ago. 2008. Disponível em: <http://www.blogcmmi.com.br/o-que-e/o-que-e-cmmi>. Acesso em: 12 ago. 2013.

TAUB JÚNIOR, Adilson. **Rational Unified Process - RUP.** [S.l.]: 2009. Disponível em: <http://www.baguete.com.br/artigos/731/adilson-taub-junior/04/11/2009/rational-unified-process-rup>. Acesso em: 23 set. 2013.

TELES, Vinícius Manhães. **Extreme Programming.** [S.l.]: [2013?]. Disponível em: <http://desenvolvimentoagil.com.br/xp/>. Acesso em: 07 out.. 2013.

TELES, Vinícius Manhães. **Manifesto Ágil.** [S.l.]: 2008. Disponível em: <http://desenvolvimentoagil.com.br/xp/manifesto_agil>. Acesso em: 27 set. 2013.

VALENTE, José Armando. **Questão do Software:** parâmetros para o desenvolvimento de software educativo**.** [S.l.]: 1989. Disponível em: <http://www.nied.unicamp.br/ojs/index.php/memos/article/view/79/78>. Acesso em: 19 out. 2013.

VARGAS, Fernando. Modelo cascata. **Blog Penso TI:** para quem pensa em TI o tempo todo (Blog). [S.l.]: Fernando Vargas. 13 mar. 2011. Disponível em: <http://www.pensoti.com.br/engenharia-de-software/modelo-cascata>. Acesso em: 14 set. 2013.

YOURDON, Edward. **Declínio e queda dos analistas e dos programadores:** a salvação e os novos caminhos para a produtividade e a qualidade no desenvolvimento de software. São Paulo: Makron Books, 1995.