CENTRO UNIVERSITÁRIO EUROAMERICANO – UNIEURO
PRÓ-REITORIA E PÓS-GRADUAÇÃO, PESQUISA E EXTENSÃO
COORDENAÇÃO DE PÓS-GRADUAÇÃO LATO SENSU
MBA EM ENGENHARIA DE *SOFTWARE*

MURILO SILVA ANDRADE SOUZA

NAYARA SOUZA DUARTE

ROSANA OLIVEIRA RAMOS DA COSTA

QUALIDADE DE *SOFTWARE* COM O RUP

Brasília, Abril /2014

MURILO SILVA ANDRADE SOUZA
NAYARA SOUZA DUARTE
ROSANA OLIVEIRA RAMOS DA COSTA

QUALIDADE DE *SOFTWARE* COM O RUP

Trabalho de conclusão de Curso apresentado como pré-requisito parcial para a conclusão do curso de MBA em Engenharia de *Software* do Centro Universitário Euroamericano – Unieuro.

Orientador: Prof.ª Dra. Edna Dias Canedo

Brasília, Abril /2014

MURILO SILVA ANDRADE SOUZA
NAYARA SOUZA DUARTE
ROSANA OLIVEIRA RAMOS DA COSTA

QUALIDADE DE *SOFTWARE* COM O RUP

Esta monografia foi julgada adequada à obtenção do grau de Especialista em MBA em Teste de *Software* à Distância e aprovada em sua forma final pelo curso de Pós-graduação Lato Sensu em MBA em Teste de *Software* à Distância do Centro Universitário UNIEURO.

Data de aprovação:

Banca Examinadora

_____

Profª. Dra. Edna Dias Canedo - Orientadora
Centro Universitário UNIEURO

_____

ProfºMsc. Cleber Machado Ortiz
Centro Universitário UNIEURO

_____

ProfºMsc. Leôncio Regal Dutra
Centro Universitário UNIEURO

RESUMO

A qualidade está cada vez sendo mais difundida dentro das grandes empresas, com propósitos de melhorar sua gestão e as negociações com seus clientes.

A ausência de uma metodologia pode ser uma das causas da baixa qualidade na maioria dos projetos de *software*. Sendo assim, a avaliação da qualidade está diretamente relacionada com a qualidade de processos e metodologias utilizadas.

Em questão, estaremos abordando como atingir qualidade no desenvolvimento de *software* utilizando como processo de engenharia o RUP, metodologia formal composta de documentação, que mantém o histórico dos artefatos gerados e garantindo o gerenciamento de mudanças. Seu método interativo e incremental com desenvolvimento baseado em procedimentos.

A qualidade de *software* não pode ser avaliada de maneira isolada, confiabilidade, segurança, robustez e práticas deste processo de engenharia serão exemplificadas para melhor entendimento.

Palavras-Chaves: Qualidade de *Software*; RUP, Processo Unificado *Rational*.

ABSTRACT

The quality is increasingly being more widespread within large companies, for purposes of improving its management and negotiations with customers.

The absence of a methodology may be a cause of poor quality in most software projects. Therefore, the quality assessment is directly related to the quality of processes and methodologies.

Concerned, we will be addressing how to achieve quality in software development as an engineering process using RUP methodology consists of formal documentation, which keeps track of the artifacts generated and ensuring change management. His interactive and incremental method based on procedures development.

The software quality can not be evaluated in isolation, reliability, security, robustness and engineering practices of this process will be exemplified for better understanding .

Key Words: Software Quality; RUP, Rational Unified Process.

LISTA DE ABREVIATURAS

ABNT	Associação Brasileira de Normas Técnicas
ISO	*International Organization for Standardization*
IEC	*International Eletrotechnical Committee*
NBR	Norma Brasileira de Regulamentação

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Qualidade é um termo que pode ter diferentes interpretações. Pode ser definida pela perspectiva do produtor e pela perspectiva do cliente. Pela perspectiva do produtor, qualidade significa conformidade com as especificações (ou a ausência ou a variação dessas especificações). Um fabricante de relógios de pulso, por exemplo, poderia incluir uma especificação de confiabilidade que requeira que 99,995% dos relógios não atrasarão ou adiantarão mais de um segundo por mês. Testes simples permitirão ao fabricante medir precisamente essas especificações. (LAUDON, 1999).

Segundo Pressman (1995, p723) *apud* Philip Crosby [CRO79], em seu destacado livro sobre qualidade, discute essa situação:

> *O problema da administração da qualidade não é aquilo que as pessoas não sabem sobre ela. O problema é aquilo que elas sabem. [...]*
> *Nesse sentido, a qualidade tem muito em comum com o sexo. Todos se interessam por ele (sob certas condições, é claro). Todos acham que o entendem. (Ainda que não queiram explicá-lo.) Todos acham que a execução é apenas uma questão de seguir as inclinações naturais. (Afinal de contas, já nos saímos bastante bem!) E, é claro, a maioria das pessoas acha que os problemas nessas áreas são causados por outras pessoas. (Bastava que elas tivessem tempo para fazer as coisas corretamente).*

Qualidade de uma forma geral é a soma das características e critérios de um produto ou serviço que executam suas práticas para satisfazer às partes envolvidas.

Segundo a definição de Pressman, (2006), qualidade de *software* é a conformidade a requisitos funcionais e de desempenho que serão claramente declarados, e expostos a padrões de desenvolvimento claramente documentados, e a características implícitas que são esperadas de todo *software* desenvolvido por profissionais, ou seja, um *software* que atenda as necessidades do cliente com a capacidade de concentrar os requisitos funcionais e eficiência de forma harmônica.

1.1 Formulação do problema

Com a concorrência do mercado internacional de produtos e serviços de *software*, a qualidade é ponto estratégico quando nos referimos a competir no mercado de desenvolvimento de *Software.* Prazos estourados, baixa produtividade, custos altos e

qualidade deficiente parece ser um fantasma que novas tecnologias não são capazes de solucionar.

Diante dessa realidade porque existe tão pouco investimento em qualidade? Podemos concluir que o problema chega a ser cultural. A visão deturpada dos empresários de que esse tipo de investimento gera custos altos à empresa é errônea e é agravada pela limitada cultura de informática dos usuários e clientes, que tendem a pressionar por prazos completamente fora da realidade.

Mudanças são necessárias e causam, inicialmente, uma baixa na produtividade devido à curva de aprendizado. Mas a institucionalização de processos e metodologias voltadas a qualidade de produtos e serviços de *software* a médio e longo prazo trazem retornos satisfatórios.

1.2 Objetivos Gerais

O objetivo deste trabalho é especificar as diversas formas de avaliação da qualidade dos produtos de *software*, utilizando como base as características dos modelos do RUP.

Os objetivos específicos do trabalho são:

a) descrever a importância da qualidade dentro dos produtos de *software*;

b) como o RUP pode colaborar com a qualidade dos *softwares*.

1.3 Objetivos Específicos

Este trabalho tem como objetivo específico descrever a importância da qualidade e que é possível alcançá-la com a qualidade do processo, ou seja, a qualidade do produto está vinculada a qualidade do processo de produção.

Detalhando estes objetivos temos como ênfase no trabalho:

- conformidade a (PRESSMAN, 2002):
  - requisitos funcionais e de desempenho;
  - padrões e convenções de desenvolvimento pré-estabelecidos;
  - atributos implícitos que todo *software* desenvolvido profissionalmente, deve possuir;
- o RUP como um produto de processo que oferece uma estrutura de processo customizável para a engenharia de *software* para melhorar a qualidade do produto final.

1.4 Justificativa

Com os avanços da tecnologia de informação, cada vez mais as empresas se tornam dependentes de sistemas de informação que atendam, de forma eficaz e veloz, às suas necessidades.

Considerando a importância inquestionável destes sistemas que possibilitam a interconexão entre todos aos processos administrativos, operacionais e estratégicos, de qualquer organização, esses sistemas devem atender os propósitos dos clientes e terem seu desenvolvimento dentro do orçamento e prazos estipulados.

Porém, ainda existem inúmeros projetos de desenvolvimento de software que são iniciados e não chegam ao fim, pior ainda, outros são terminados consumindo prazos e orçamentos bem acima do que foi estipulado no início do projeto, além de serem considerados produtos de péssima qualidade, isto é, não estão de acordo com as especificações do cliente.

A ausência de uma metodologia durante o desenvolvimento do sistema pode ser a causa da baixa qualidade, ou seja, a qualidade do sistema está diretamente relacionada com a metodologia utilizada durante o desenvolvimento.

1.5 Metodologia de Desenvolvimento

No trabalho será realizada uma análise de conteúdo de livros, artigos e sites de diversos autores, para evidenciar como a o processo de desenvolvimento de *software* influencia na qualidade do produto final que deverá ser entregue, destacando todos os aspectos de cada fase deste processo, desde o levantamento de das necessidades do usuário até a entrega do *software* com qualidade para atender seus usuários.

1.6 Estrutura do Trabalho

Este trabalho é composto de 7 capítulos, abrangendo fundamentos de qualidade de *software* com RUP.

O capítulo 1 apresenta uma introdução ao trabalho, alguns conceitos, os objetivos e a organização do texto.

O capítulo 2 apresenta o conceito de desenvolvimento de *software* de um modo geral, além de aspectos importantes na qualidade em *software,* segundo alguns autores.

O capítulo 3 apresenta a norma ISO/IEC 9126 com seu Modelo de Qualidade.

O capítulo 4 apresenta a importância da qualidade de *software* como fundamento principal da eficiência.

O capítulo 5 apresenta o RUP com seus principais conceitos e seus objetivos dentro da qualidade de *software*.

O capítulo 6 apresenta a qualidade dos *softwares* utilizando o RUP apresentando os artefatos de cada fase.

Por fim, no capítulo 7 concluímos o trabalho expondo a importância do RUP na qualidade dos *softwares*.

2 DESENVOLVIMENTO DE *SOFTWARE*

Focando no desenvolvimento de *software*, Ian Sommerville (2002) define um processo de *software* como um conjunto de atividades que leva à produção de um produto de *software*. Roger S. Pressman (2006) define processo de *software* como um arcabouço para as tarefas que são necessárias para construir *software* de alta qualidade. Wilson de Paula Filho (2009) faz uma analogia interessante, para ele, processo é uma receita a ser seguida.

Processos de *software*s são complexos e como todos os processos intelectuais e criativos dependem de julgamento humano. A existência de um processo de *software* não garante que o *software* será entregue no prazo, de que ele irá satisfazer as necessidades do cliente, ou exibirá os atributos arquiteturais que manterão as características de qualidade em longo prazo. Um processo deve ser acoplado a uma sólida prática de engenharia de *software* e deve ser avaliado para garantir que satisfaça a um conjunto de critérios básicos de processo que demonstram ser essenciais para uma engenharia de *software* bem sucedida (PRESSMAN, 2006).

A engenharia de *software* surgiu num contexto onde a crise de *software* se apresentava e, decorrente disso, sistemática de trabalho mais consistentes e formais, inspiradas na engenharia, foram concebidas para solucionar os problemas que tendiam ser, cada dia, maiores e mais complexos e que acompanhavam a comunidade de desenvolvedores ao longo dos anos, de forma crônica (PRESSMAN, 1995).

A partir deste cenário, surgiu a necessidade de tornar o desenvolvimento de *Software* como um processo planejado e padronizado, para que as necessidades fossem atendidas e os gastos controlados.

Entretanto, a crise do *Software* ainda perdura, onde, mesmo depois de anos, ainda existem características da época da crise, como projetos atrasados, erros de estimativa de custos e de tempo, que tornam o processo, ainda que sistematizado, passível de muitos erros.

A maior prioridade é satisfazer o cliente, a partir de entregas de produtos de *software* de efetivo valor em tempo hábil e continuamente; Acatar as necessidades de mudanças em qualquer estágio do processo de desenvolvimento, pois o *software* deve prover efetiva vantagem competitiva ao cliente.

A engenharia de *software* surgiu com o intuito de solucionar problemas que, pela sua magnitude, foram denominados de crise de *software*. Independentemente de ser uma crise ou problema crônico, o fato é que as dificuldades persistem (PRESSMAN, 2002).

Para chegar a tal ponto, os processos foram desenvolvidos, porém nunca um deles trará satisfação total somada com qualidade inquestionável, seja em qualquer parte do projeto ou iteração de um processo.

2.1 Aspectos importantes das definições de qualidade de *software*

Os requisitos de *software* são a base a partir da qual a qualidade é medida. A falta de conformidade com os requisitos significa falta de qualidade;

Um requisito de *software* descreve o que é requerido para que o sistema cumpra o seu objetivo.

Engenharia de Requisitos é o processo de descobrir, analisar, documentar e verificar as funções e restrições do sistema (SOMMERVILLE, 2004).

Para que os requisitos do *software* sejam atingidos é necessário, definir o que o sistema deve fazer, e não como ele deve ser implementado, são organizados e acordo com os diferentes subsistemas que constituem o sistema.

Para estabelecer o que o sistema deve fazer é necessário que dentro do processo de desenvolvimento de sistemas, a atividade engenharia de requisitos produz um documento que retrata de forma gera-lo que o sistema deve fazer. Compreender a natureza dos problemas pode ser muito difícil, especialmente se o sistema for novo (SOMMERVILLE, 2004).

Consequentemente é difícil estabelecer com exatidão o que o sistema vai fazer. As descrições das funções e das restrições são os requisitos para o sistema; e o processo de descobrir, analisar, documentar e verificar essas funções e restrições é chamado de engenharia de requisitos (PRESSMAN, 2006).

Outro aspecto de definições que devem ser verificados é a definição de um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o *software* passa pelo trabalho de engenharia. Caso os critérios não sejam seguidos, o resultado quase que seguramente será a falta de qualidade.

Para que um *software* seja desenvolvido de forma consistente, é preciso aliar boas práticas da engenharia de *software* com um robusto e eficiente processo de desenvolvimento. Diferentes tipos de sistemas necessitam de diferentes processos de desenvolvimento. Por exemplo, um *software* de tempo real de uma aeronave deve ser completamente especificado antes do inicio do desenvolvimento, enquanto que um sistema de comércio eletrônico a especificação e o desenvolvimento do *software* podem ser conduzidos paralelamente. O uso

de um processo de *software* inadequado pode reduzir a qualidade ou a utilidade do produto de *software* a ser desenvolvido e/ou aumentar os custos de desenvolvimento. Este fato leva as organizações que produzem *software* a usar processos de desenvolvimento que sejam eficientes e que atendam plenamente suas necessidades (SOMMERVILE, 2007).

Existe um conjunto de requisitos implícitos que frequentemente não são mencionados na especificação, são os requisitos não funcionais que Pressman (2002), define como: Fatores de qualidade de *software* que podem ser medidos de forma indireta, ou como características implícitas que são esperadas de todo *software* profissionalmente desenvolvido. Por exemplo, o desejo de uma boa Integridade no acesso ao Sistema.

2.2 Qualidade em *Software*

Segundo Inthurn (2001), o desenvolvimento de *software* com qualidade é um assunto amplo, complexo e ainda muito discutido. São vários os fatores que precisam ser considerados para obtermos um resultado satisfatório. No entanto, é sempre bom lembrar que o principal indicador de qualidade no desenvolvimento de qualquer produto, incluindo o *software*, é a satisfação do cliente.

Podemos então, definir qualidade de *software* como um conjunto de propriedades a serem satisfeitas em determinado grau, de modo que o *software* satisfaça as necessidades de seus usuários segundo (ROCHA, 2001).

Neste contexto, pessoas com diferentes interesses sobre um produto têm diferentes visões sobre o conceito de qualidade. Clientes usualmente consideram que o *software* tem qualidade se possui características que atendam suas necessidades. Desenvolvedores usualmente veem a qualidade através das medidas de suas propriedades que são comparadas com indicadores de qualidade preestabelecidos (CÔRTES e CHIOSSI, 2001).

Um cliente estará satisfeito quando um *software* adquirido por ele atender suas necessidades sem que para isso seja necessário um conhecimento abrangente na área de *software*, afinal ele precisa de conhecimento abrangente na sua área de atuação, e não em *software*. Porém, nem sempre é fácil escolher um *software* que satisfaça todas as suas necessidades apenas lendo suas características.

Desta forma, estas novas exigências fizeram com que as empresas adotassem novas filosofias para “sobreviver” no mercado mais competitivo que surgiu. Tendo o Programa Brasileiro da Qualidade e Produtividade papel importante neste processo.

Segundo Weber & Rocha (2001), o Programa Brasileiro da Qualidade e Produtividade teve uma primeira fase que foi de 1990 até 1995, e foi nesta fase, mais precisamente em 1º de junho de 1993, no âmbito do PBQP, que foi criado o Subcomitê Setorial da Qualidade e Produtividade em *Software* (PBQP/SSQP-SW). Sendo o termo de referência do PBQP/SSQPSW composto de:

a) diagnóstico do setor de *software* em relação à qualidade e produtividade;
b) análise das tendências nacionais e internacionais da qualidade e produtividade em *software*;
c) objetivo, estratégias e ações para soluções dos problemas que influenciam na obtenção de padrões internacionais de qualidade e produtividade em software.

Mas, para que este termo de referência seja colocado em prática não é necessário somente desejar e, sim que se minimizem o número de defeitos, que se crie mecanismos para garantir prazos e custos, garantirem o uso do produto no mercado e melhorar qualidade de novas versões, além de melhorar a qualidade de novos produtos com base na experiência adquirida com os desenvolvimentos anteriores.

Portanto, segundo Rocha (2001), não é suficiente desejar que o *software* a ser produzido atinja um nível adequado de qualidade. Para que este nível seja alcançado, é necessário buscá-lo desde o início do desenvolvimento, uma vez que níveis satisfatórios de qualidade só são assegurados se o grupo de desenvolvedores atuarem conscientemente e deliberadamente para atingi-los. Fernandes (1995) vai além quando menciona que esses requisitos podem evoluir com o tempo. Por isso é que este é um processo dinâmico e evolutivo.

Em face disto, a informática que nos apresenta um novo mundo, quebrando todas as barreiras de comunicação e modificando a própria estrutura da indústria e do consumo (INTHURN, 2001), torna-se uma grande aliada para o processo de globalização. Por isso, seria impossível para as empresas que trabalham com o desenvolvimento de *software* ficassem fora deste processo. E sendo a qualidade uma das principais exigências para este novo mercado globalizado; quem possuir produto com qualidade e baixo custo, estará na frente daqueles que não acreditarem nesta “revolução”.

Segundo Ferreira (2003), havendo liberdade de escolha, os clientes das organizações públicas e privadas fazem uma série de exigências. Tudo aquilo que estão acostumados a obter torna-se, para eles, qualidade obrigatória. Portanto, para conquistar clientes, diante da competição atual, é preciso encantá-los e seduzi-los. Qualidade intrínseca, preço acessível, flexibilidade, baixo custo de manutenção, valor de revenda, prazo de entrega, rapidez,

variedade de opções, cordialidade no atendimento, boas condições de pagamento, imagem no mercado, segurança pessoal e ambiental são algumas entre as muitas possibilidades exploradas como vantagens competitivas.

Neste contexto, é de extrema importância conhecer as diferentes visões de um *software* para conquistar estes clientes. E através da norma ISO/IEC 9126, tem-se o conhecimento da visão do gerente de desenvolvimento, do desenvolvedor e a do usuário; sendo estas úteis para a tomada de decisões no processo de desenvolvimento de um *software*.

Nos capítulos a seguir serão descritas estas visões segundo a ISO/IEC 9126.

3 A NORMA ISO/IEC 9126

A norma ISO/IEC 9126 trata da tecnologia de informação, com base na avaliação de produtos de *software*, das características de qualidade e diretrizes para o seu uso.

A série de normas ISO/IEC 9126 propõe um modelo de qualidade do produto de *software* que é dito como referência para avaliação da qualidade. A primeira parte NBR ISO/IEC 9126-1 está subdividido em duas partes: modelo de qualidade para características externas e internas e modelo de qualidade para qualidade em uso (ROCHA, 2001, 116p).

Segundo Sergio a maior contribuição da norma NBR ISO/IEC 9126-1 é ter definido e estabelecido um modelo teórico de qualidade. Contudo, para a aplicação prática desse modelo dentro de um processo de avaliação de produto de *software*, ela deve ser usada em conjunto com a série NBR ISO/IEC 14598 (2004, 38p.).

3.1 ISO/IEC 9126-1: Modelo de qualidade

Como descrito anteriormente, a norma ISO/IEC 9126-1 é formada por dois Modelos de Qualidade: o Modelo de Qualidade Interna e Externa e o Modelo de Qualidade em Uso. Esses dois modelos e suas respectivas características serão detalhados a seguir.

3.1.1 Modelo de qualidade para qualidade interna e externa

O modelo de qualidade externa e interna especifica seis características de qualidade que são, por sua vez, subdivididas em subcaracterísticas. As subcaracterísticas podem ser desdobradas em mais níveis que caracterizam os atributos de qualidade. As métricas internas e externas aplicam-se, em geral, ao nível dos atributos de qualidade (ROCHA, 2001, 116p).

Os dois primeiros níveis estão bem definidos e é deixado a cargo do usuário da norma identificar quais atributos são relevantes no produto que será avaliado (KOSCIANSKI, 1999, 25p).

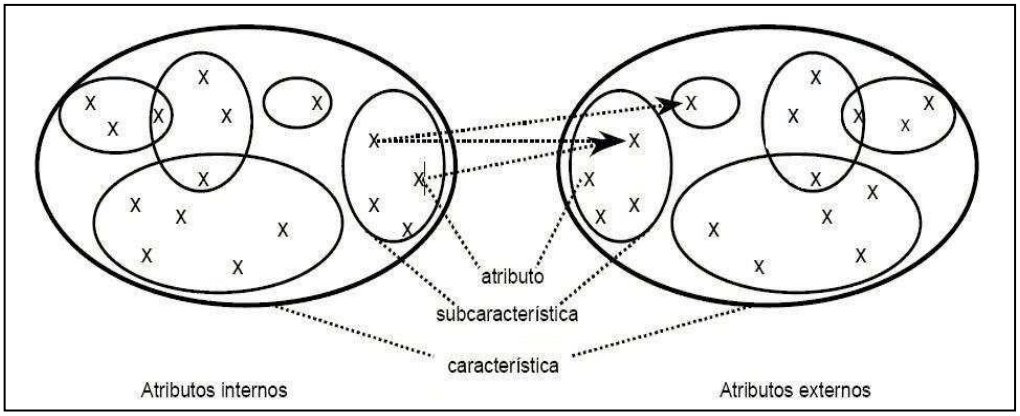

Figura 3-1 - Características, Subcaracterísticas e Atributos de Qualidade de *software*

Fonte: NBR ISO/IEC 9126 (2003)

Qualidade interna é a totalidade de características do produto de *software* na visão interna. Ela é usada para especificar as propriedades do produto de *software* intermediário.

Analogamente, a qualidade externa é a totalidade de características do produto de *software* do ponto de vista externo. É a qualidade que normalmente é avaliada quando o produto de *software* final está sendo testado.

Como é possível visualizar na Figura 3-2, o modelo de qualidade para qualidade interna e externa possui definições de seis características básicas que um produto de *software* deve ter para ser considerado um *software* de qualidade: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

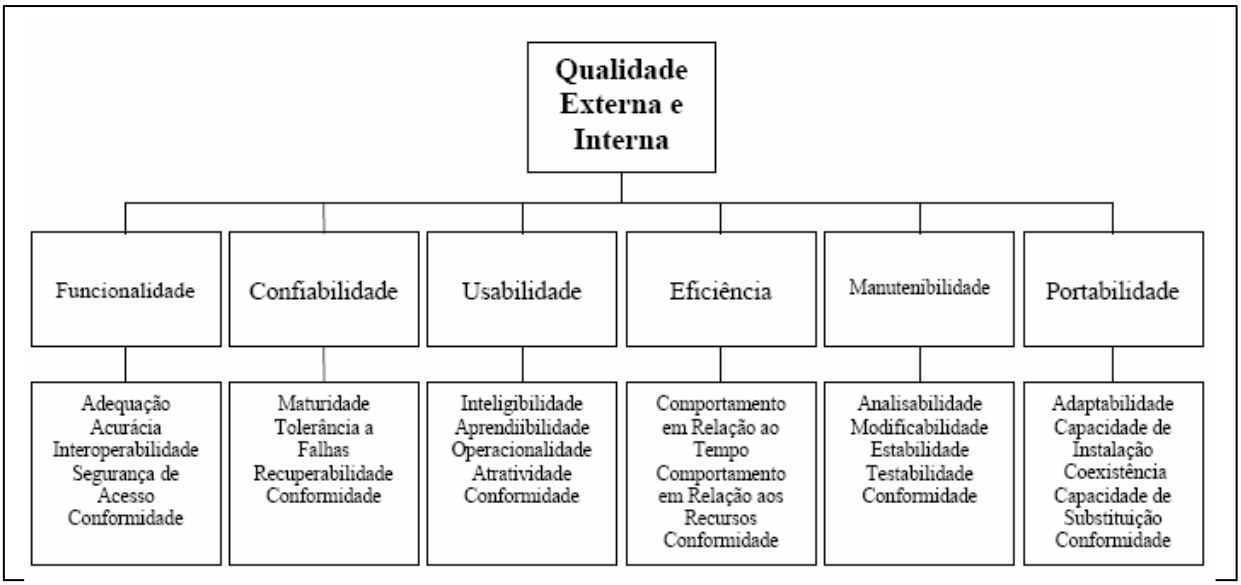

Figura 3-2 - Características e Subcaracterísticas de Qualidade Externa e Interna

Fonte: Machado, M. P.; Souza, S. F (2011). *Métricas e Qualidade de Software*

A seguir temos as definições que são atribuídas para cada característica e para cada subcaracterística do *software* que influencia a característica de qualidade (NBR ISO/IEC 9126-1, 2003, 7p).

a. Funcionalidade: Capacidade do produto de *software* de prover funções que atendam às necessidades explícitas e implícitas, quando o *software* estiver sendo utilizado sob condições especificadas.

- **Adequação:** Capacidade do produto de *software* de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados.
- **Acurácia:** Capacidade do produto de *software* de prover, com grau de precisão necessário, resultados ou efeitos corretos ou conforme acordados.
- **Interoperabilidade:** Capacidade do produto de software interagir com um ou mais sistemas especificados.
- **Segurança de acesso:** Capacidade do produto de *software* de proteger informações e dados, de forma que pessoas ou sistemas não autorizados não possam lê-los nem modificá-los e que não seja negado acesso às pessoas ou sistemas autorizados.
- **Conformidade relacionada à funcionalidade:** Capacidade do produto de *software* de estar de acordo com normas, convenções ou regulamentações previstas em leis e prescrições similares relacionadas à funcionalidade.

b. Confiabilidade: Capacidade do produto de *software* de manter um nível de desempenho especificado, quando usado em condições especificadas.

- **Maturidade:** Capacidade do produto de *software* de evitar falhas decorrentes de defeitos no *software*.
- **Tolerância a falhas:** Capacidade do produto de *software* de manter um nível de desempenho especificado em casos de defeitos no *software* ou de violação de sua interface especificada.
- **Recuperabilidade:** Capacidade do produto de *software* de restabelecer seu nível de desempenho especificado e recuperar os dados diretamente afetados no caso de uma falha.
- **Conformidade relacionada à confiabilidade:** Capacidade do produto de *software* de estar de acordo com normas, convenções ou regulamentações relacionadas à confiabilidade.

c. Usabilidade: Capacidade do produto de *software* de ser compreendido, aprendido, operado e atraente ao usuário, quando usado sob condições especificadas.

- **Inteligibilidade:** Capacidade do produto de *software* de possibilitar ao usuário compreender se o *software* é apropriado e como ele pode ser usado para tarefas e condições de uso específicas.
- **Apreensibilidade:** Capacidade do produto de software de possibilitar ao usuário aprender sua aplicação.
- **Operacionalidade:** Capacidade do produto de *software* de possibilitar ao usuário operá-lo e controlá-lo.
- **Atratividade:** Capacidade do produto de *software* de ser atraente ao usuário.
- **Conformidade relacionada à usabilidade:** Capacidade do produto de *software* de estar de acordo com normas, convenções, guia de estilo ou regulamentações relacionadas à usabilidade.

d. Eficiência: Capacidade do produto de *software* de apresentar desempenho apropriado, relativo à quantidade de recursos usados, sob condições especificadas.

- **Comportamento em relação ao tempo:** Capacidade do produto de *software* de fornecer tempos de resposta e de processamento, além de taxas de transferência, apropriados, quando o *software* executa suas funções, sob condições estabelecidas.
- **Utilização de recursos:** Capacidade do produto de *software* de usar tipos e quantidades apropriados de recursos, quando o software executa suas funções sob condições estabelecidas.
- **Conformidade relacionada à eficiência:** Capacidade do produto de *software* de estar de acordo com normas e convenções relacionadas à eficiência.

e. Manutenibilidade: Capacidade do produto de *software* de ser modificado**.**

- **Analisabilidade:** Capacidade do produto de *software* de permitir o diagnóstico de deficiências ou causas de falhas no *software*, ou adaptações do *software* devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais.
- **Modificabilidade:** Capacidade do produto de *software* de permitir que uma modificação especificada seja implementada.
- **Estabilidade:** Capacidade do produto de *software* de evitar efeitos inesperados decorrentes de modificações no *software*.

- **Testabilidade:** Capacidade do produto de *software* de permitir que o *software*, quando modificado, seja validado.
- **Conformidade relacionada à manutenibilidade:** Capacidade do produto de software de estar de acordo com normas ou convenções relacionadas à manutenibilidade.

f. **Portabilidade:** Capacidade do produto de *software* de ser transferido de um ambiente para outro.

- **Adaptabilidade:** Capacidade do produto de software de ser adaptado para diferentes ambientes especificados, sem necessidade de aplicação de outras ações ou meios além daqueles fornecidos para essa finalidade pelo *software* considerado.
- **Capacidade para ser instalado:** Capacidade do produto de *software* para ser instalado em um ambiente especificado.
- **Coexistência:** Capacidade do produto de *software* de coexistir com outros produtos de *software* independentes, em um ambiente comum, compartilhando recursos comuns.
- **Capacidade para substituir:** Capacidade do produto de *software* de ser usado em substituição a outro produto de software especificado, com o mesmo propósito e no mesmo ambiente.
- **Conformidade relacionada à portabilidade**: Capacidade do produto de *software* de estar de acordo com normas ou convenções relacionadas à portabilidade.

3.1.2 Modelo de qualidade para qualidade em uso

Qualidade em uso é a visão de qualidade sob a perspectiva do usuário e é medido pelo efeito do uso do *software*. O modelo de Qualidade para Qualidade em Uso faz a avaliação de quanto o usuário pode atingir seus objetivos em um ambiente, sem medir as propriedades do produto de *software*. Esse modelo possui quatro características de qualidade: Efetividade, Produtividade, Segurança e Satisfação, como destacado na Figura 3-3.

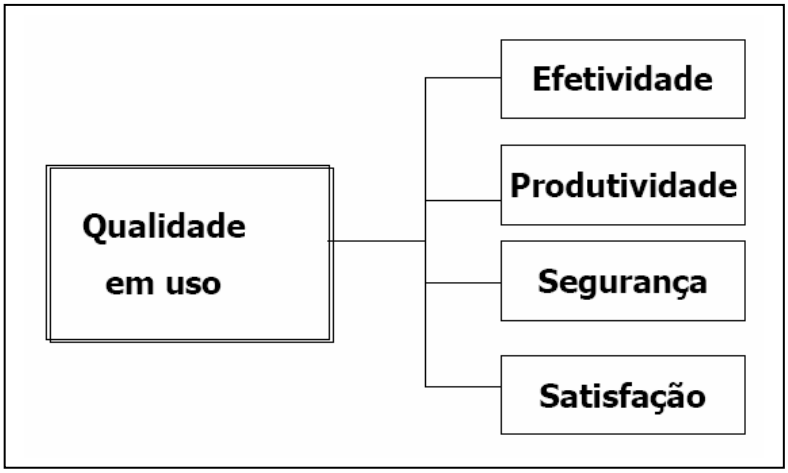

Figura 3-3 - Características de Qualidade em Uso

Fonte: PALUDO, M. (2010) Qualidade de Produto de *Software*

Cada uma das quatro características de Qualidade em Uso será descrita a seguir:

- **efetividade**: capacidade do produto de *software* de permitir ao usuário atingir metas específicas com acurácia e completude, em um contexto de uso específico;
- **produtividade**: capacidade do produto de *software* de permitir que seus usuários empreguem quantidade adequada de recursos em relação à efetividade alcançada em um contexto de uso específico;
- **segurança**: capacidade do produto de *software* de apresentar níveis aceitáveis de riscos de danos a pessoas, negócios, *software*, propriedade ou ambiente em um contexto de uso específico;
- **satisfação**: capacidade do produto de *software* de satisfazer usuários em um contexto de uso específico.

4 QUALIDADE DE SOFTWARE: UMA QUESTÃO DE EFICIÊNCIA

Você sabia que, a grande maioria das empresas de *software* no Brasil gasta 70% do tempo de desenvolvimento corrigindo erros, ao invés de inovar e desenvolver novas soluções?

É um trabalho de reescrever códigos ao invés de criar novos códigos, inovar, criar novas soluções. Existem empresas que já estão na versão 5 de seu *software*, mas tem clientes que ainda usam a versão 3, porque os clientes morrem de medo de atualizar, por causa dos históricos de erros em novas versões – eles preferem os problemas já conhecidos. Empresas que levam o *software* ao cliente e uma tela não funciona, ou uma correção de um problema que afetou outro lugar no sistema e uma infinidade de questões, aqui vamos ilustrar apenas algumas:

4.1 Não existem requisitos ou documentação

Geralmente desenvolvedores não fazem a tarefa de casa levantando os requisitos de *software*, já começam a escrever o código conforme é pedido. Ou, pior, não fazem as documentações necessárias de análise antes do desenvolvimento de um *software*. Isso causa diversos problemas:

- quando o desenvolvedor desconhece os requisitos, ele provavelmente voltará a reescrever um código que já foi escrito, pela falta de análise, desordem, descaso por não levantar os requisitos;
- às vezes pode até haver requisitos, mas numa equipe cada desenvolvedor pode interpretar os requisitos de uma forma diferente;
- requisitos mal levantados ou mal escritos por falta de conhecimento ou experiência;
- falta de uma documentação do *software*;
- documentação mal elaborada.

Quando há a falta de documentação o serviço de *software* fica complicado em diversos sentidos: há uma grande dificuldade de encontrar mão de obra especializada, e quando encontram, devido à falta de documentação, os contratados levam meses para conhecer o *software* da empresa, afinal, o conhecimento está na cabeça de alguns; é difícil fazer teste de *software* para garantir um menor risco de erros; o desenvolvimento de *software*

é comprometido aumentando as chances de erros do *software*. Estes são apenas alguns exemplos.

4.2 Não existe a fase de projeto de *software*

Simplesmente levantam-se rapidamente os requisitos e inicia o desenvolvimento, sem uma analise e projeto. Isso leva a falta de documentação e inicio de vários problemas de, no futuro, ter que reescrever códigos com erro, além de dificultar para a equipe de testes a análise do *software*.

4.3 Controle de mudanças e de versões inadequadas (ou inexistentes)

Não há um controle do que foi mudado, um muda o banco de dados sem comunicar e interferem outros módulos do *software*. Não há uso de controle de versões. O que é um erro. Há diversas ferramentas que fazem este controle, como o Sistema de Controle de Versão - **CVS**.

4.4 Foco na entrega

Muitos pensam em entregar o mais rápido o *software*, sem se preocupar com a qualidade. Isso faz o *software* voltar com diversos erros encontrados, muitas vezes pelos clientes, o que é péssimo. É melhor colocar um prazo real e entregar com qualidade, que correr para entregar rápido. Há empresas que eliminam a análise de documentação do projeto em nome de "produtividade", mas esta suposta "produtividade" é apenas ilusória.

4.5 Inexistência de um time de testes

Geralmente quem faz os testes são os desenvolvedores. Isso é um erro. Eles seguem uma lógica que é diferente da lógica de usuários finais. Ou quando tem uma equipe de testes, é a secretária, o *office-boy*, ou outros funcionários da empresa que não fazem um teste correto. Ou a falta de ferramentas de automação de testes. Ou o teste é realizado pelos próprios clientes, o que é pior.

4.6 Time de testes focado em testes superficiais

Quando há, como foi citado no item anterior, não utilizam ferramentas de automação de testes, de ferramentas de gestão de testes, de gestão de defeitos etc. Fazem aquele teste de usuário, mesmo assim superficial, não fazem teste do banco de dados, do desempenho do sistema, de análise do código, não catalogam os defeitos em níveis de defeitos, não criam regras de mensagens de erros para ajudar ao usuário a contornar os possíveis erros etc. Hoje montar uma equipe de testes é importante, se estuda muito teste de *software*, são raros os profissionais especializados e são muito bem remunerados.

5 PROCESSO UNIFICADO DA RATIONAL (RATIONAL UNIFIED PROCESS – RUP)

É um processo de engenharia de *software* criado para apoiar o desenvolvimento orientado a objetos, fornecendo uma forma sistemática para se obter vantagens no uso da UML (*Unified Modeling Language*) e que fornece um conjunto de práticas. Trata-se de um processo proprietário, desenvolvido pela *Rational Software*, atualmente subsidiária da IBM. O ciclo de vida é dividido em fases sequenciais, as quais podem ser subdivididas em iterações.

Não existe uma fórmula exata para aplicação do RUP, isso dependerá de cada projeto e organização. Contudo algumas características diferenciam o RUP de outros métodos:

- **adaptação do processo**: projetos possuem tamanhos de escopo diferentes exigindo uma constante adequação;
- **balanceamento das prioridades dos envolvidos**: um grande desafio durante o desenvolvimento do *software*, mas garante alinhamento do negócio;
- **colaboração e comunicação entre as equipes**: com os sistemas sendo arquitetado em um lado e o código sendo desenvolvido de outro, a comunicação efetiva e unificada é um fator de sucesso;
- **demonstração de valor iterativamente**: para um processo de desenvolvimento orientado ao negócio, demanda que o time de projeto demonstre o retorno do investimento em uma abordagem repetitiva ao longo do projeto;
- **elevação no nível de abstração**: uma abordagem de sucesso para sanar as complexidades é aprimorar o uso de padrões e serviços, reutilizando componentes em vez de construir o *software* do zero;
- **foco contínuo na qualidade**: desde o levantamento de requisito e construção da documentação. A qualidade do sistema é afetada pela sua arquitetura. A inspeção de código e os testes unitários garantem que o código está de acordo com os padrões estabelecidos. Os testadores testam baseados nos cenários de casos de uso, ou com requisitos não funcionais. A responsabilidade de encontrar erros deve ser compartilhada entre todos da equipe.

Abaixo a Figura 5-1 demonstra os princípios do RUP dentro de uma linha continua de processos.

Figura 5-1 – Princípios do RUP

Fonte: *Autoria Própria (Murilo Andrade)*

5.1 Conceitos do RUP

O principal objetivo do RUP é atender as necessidades dos usuários garantindo uma produção de *software* de alta qualidade que cumpra um cronograma e um orçamento previsíveis.

O RUP organiza o desenvolvimento do ciclo de vida do *software* em quatro fases, onde cada fase tem um papel fundamental para que o objetivo seja cumprido e cada fase possui o foco em uma parte do desenvolvimento que são:

- **iniciação ou concepção**: ênfase no escopo do sistema;
- **elaboração**: ênfase na arquitetura;
- **construção**: ênfase no desenvolvimento;
- **transição**: ênfase na implantação.

Segundo Kroll e Kruchten (2003), o RUP tem como cinco elementos principais os papéis, atividades, artefatos, fluxos de trabalho e disciplinas.

Um papel (ou perfil) define o comportamento e as responsabilidades de um determinado indivíduo ou grupo de indivíduos trabalhando como uma equipe.

Uma atividade é uma unidade de trabalho que um indivíduo executa quando está exercendo um determinado papel e produz um resultado importante para o contexto do projeto.

Um artefato é um pedaço de informação que é produzido, modificado ou utilizado em um processo.

Um fluxo de trabalho é uma sequência de atividades que são executadas para a produção de um resultado valioso para o projeto (KROLL e KRUCHTEN, 2003).

Uma disciplina é uma coleção de atividades relacionadas que fazem parte de um contexto comum em um projeto (MARTINS, 2007). O RUP possui nove disciplinas, divididas em disciplinas do processo e de suporte. As disciplinas de processo são: modelagem de negócios; requisitos; análise e projeto; implementação; e teste e distribuição. As de suporte são: configuração e gerenciamento de mudanças; e gerenciamento de projeto e ambiente (KRUCHTEN, 2003).

Segundo Martins (2007), em sua arquitetura geral, o RUP possui duas dimensões, conforme mostra a Figura 5-2.

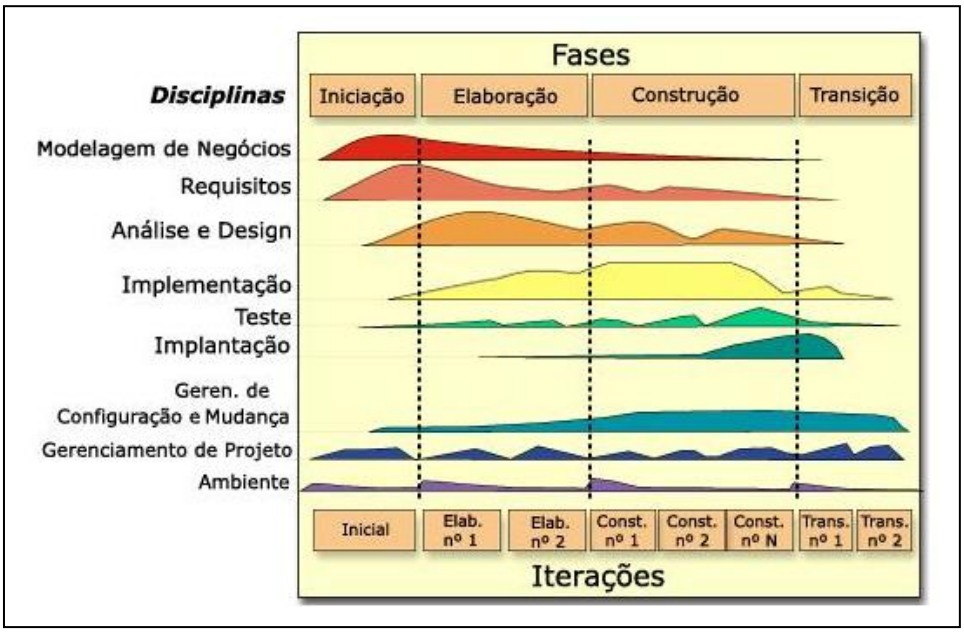

Figura 5-2 - Arquitetura Geral RUP

Fonte: *IBM_Rational_Unified_Process*[1]

O eixo horizontal representa o tempo e mostra os aspectos do ciclo de vida do processo à medida que se desenvolve. Este representa o aspecto dinâmico do processo e é expresso em termos de fases, disciplinas e marcos.

O eixo vertical representa as disciplinas, que agrupam as atividades de maneira lógica, por natureza. Este representa o aspecto estático do processo e é descrito em termos de componentes, disciplinas, atividades, fluxos de trabalho, artefatos e papéis do processo.

[1] Disponível em: < http://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process> Acessado em 08 de novembro de 2013.

Com a utilização de uma metodologia de desenvolvimento de *software* como o RUP, é possível obter:

- qualidade de *software*;
- produtividade no desenvolvimento, operação e manutenção de *software*;
- controle sobre desenvolvimento dentro de custos, prazos e níveis de qualidade desejados;
- estimativa de prazos e custos com maior precisão.

5.2 Ciclo de Vida de um Desenvolvimento de *software* com RUP

O ciclo de vida do RUP é dividido a partir das suas quatro fases (KROLL e KRUCHTEN, 2003) cada uma delas é concluída por um marco principal (MARTINS, 2007), com isso também é possível concluir que cada fase é basicamente um intervalo de tempo entre dois marcos principais.

Na Figura 5-3 abaixo, segue a ilustração das fases do RUP.

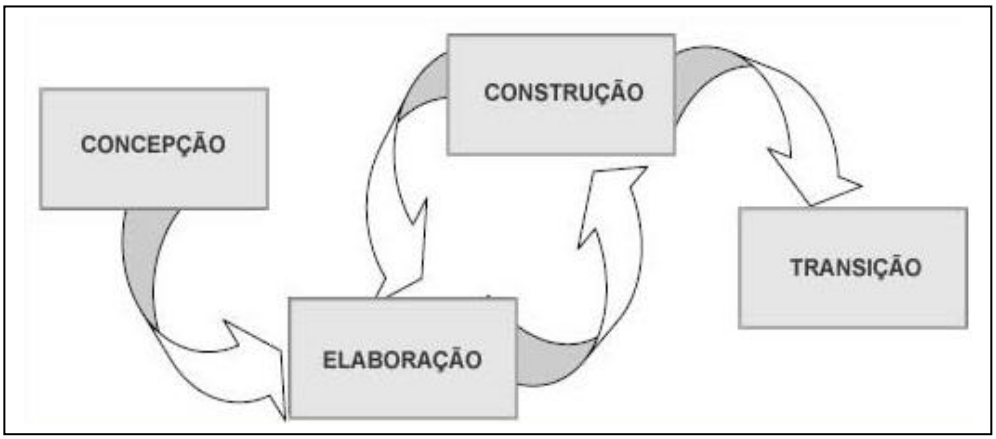

Figura 5-3 - Fases do RUP

Fonte: Info Escola – Navegando e Aprendendo[2]

Fase de Concepção / Iniciação: Requisitos de domínio na concepção de Sommerville são o que outros autores, tal como Wiegers (2003), chamam de regras de negócio. Essas regras de negócio geralmente incluem terminologia específica do domínio e fazem referência a conceitos do domínio (SOMMERVILLE, 2006). Assim, são mais facilmente capturadas na fase de modelagem conceitual. Os requisitos devem ser redigidos de modo a serem passíveis

[2] Disponível em: <**http://www.infoescola.com/engenharia-de-*software*/rup/**> Acessado em 08 de novembro de 2013.

de entendimento pelos diversos interessados (*stakeholders*). Clientes, usuários finais e desenvolvedores.

É feito um plano de projeto avaliando os possíveis riscos, as estimativas de custo e prazos, estabelecendo as prioridades, levantamento dos requisitos do sistema e preliminarmente analisá-lo. Segundo Boente (2003) afirma que "não existe projeto com risco zero". Sendo assim, a tarefa de quantificação dos riscos de um projeto, passa a ser prioridade para implementação do projeto.

Fase de Elaboração: A meta da fase de elaboração é criar a *baseline* para a arquitetura do sistema a fim de fornecer uma base estável para o esforço da fase de construção. A arquitetura se desenvolve a partir de um exame dos requisitos mais significativos (aqueles que têm grande impacto na arquitetura do sistema) e de uma avaliação de risco. A estabilidade da arquitetura é avaliada através de um ou mais protótipos de arquitetura.

Fase de Construção: Começa-se o desenvolvimento físico do *software*, produção de códigos, testes alfa. O principal objetivo desta fase é a construção do sistema de *software*, com foco no desenvolvimento de componentes e outros recursos do sistema, também é preciso esclarecer os requisitos restantes bem como gerenciar os recursos de controle de operações para otimizar custos, programações e qualidade.

Fase de Transição: Abrange a entrega do *software* ao usuário e a fase de testes. O objetivo desta fase é disponibilizar o sistema, tornando-o disponível e compreendido pelo usuário final. As atividades desta fase incluem o treinamento dos usuários finais e também a realização de testes da versão beta do sistema visando garantir que o mesmo possua o nível adequado de qualidade.

5.3 Utilização do RUP

O RUP foi desenvolvido visando dois grupos primários de usuários que dependem de sua capacidade para o desenvolvimento dos projetos:

- o primeiro grupo é composto por profissionais de desenvolvimento de *software* que trabalham como parte de uma equipe de projeto; e
- o segundo grupo é de profissionais de engenharia de processo, especificamente engenheiros de processo de *software* e gerentes.

Os profissionais de desenvolvimento de *software* podem encontrar orientação sobre o que é exigido deles nas competências e funções definidas no RUP. Um profissional que trabalha em um projeto de engenharia de *software* RUP é designado a uma ou mais funções definidas no RUP, em que cada função particiona um conjunto de tarefas e produtos de trabalho pelos quais essa função é responsável.

O RUP fornece a um profissional de desenvolvimento de *software* um ambiente de processo configurável, todavia com base em padrões, com práticas coletadas da engenharia de *software*.

As práticas recomendadas pelo RUP foram definidas de forma a aumentar as chances de sucesso de um projeto. As situações abaixo devem ser evitadas para que um *software* alcance a qualidade:

- falta de uma definição formal de um processo de gerenciamento de mudanças;
- propagação de mudanças descontroladas;
- comunicação ambígua e imprecisa;
- arquiteturas não robustas;
- ala complexidade;
- inconsistências não detectadas em requisitos e na implementação;
- testes insuficientes do sistema;
- controle subjetivo do status do projeto;
- falha ao atacar os riscos do projeto;
- automação insuficiente.

5.4 Adaptação do RUP para o projeto

As práticas do RUP é o desenvolvimento iterativo e incremental. No desenvolvimento incremental várias partes do projeto são desenvolvidas em paralelo, e integradas quando completas, já a alternativa ao desenvolvimento incremental é desenvolver todo o sistema com uma integração única.

O desenvolvimento iterativo é uma estratégia de planejamento de retrabalho em que o tempo de revisão e melhorias de partes do sistema é pré-definido.

Os usuários devem receber as liberações de versão de cada iteração, o que facilita seu retorno sobre o *software* produzido, apontando o que foi mal entendido e esclarecendo os

demais requisitos. Além disso os usuários *stakeholders* tem evidencias concretas do andamento do sistema.

Com este desenvolvimento iterativo do RUP, o *software* pode ser verificado constantemente, a verificação se dá por meio de testes do sistema. Desta forma, é um processo contínuo de avaliação qualitativa (testes executados com sucesso) e quantitativa (número de testes realizados, número de cenários ainda não testados, número de testes cujo o resultado foi bem sucedido etc.). A verificação contínua de qualidade faz com que o acompanhamento do status do projeto seja mais objetivo, as inconsistências em requisitos, projeto e implementações sejam evidenciadas, os riscos mitigados, os defeitos encontrados precocemente.

Porém não tente "executar" todo o RUP ao mesmo tempo. Adote uma abordagem para implementar. Inicie avaliando o processo existente e selecionando uma ou duas áreas-chave que você gostaria de aprimorar. Comece utilizando o RUP para aprimorar primeiro essas áreas e, depois, em iterações ou ciclos de desenvolvimento posteriores, fazer aprimoramentos incrementais em outras áreas.

Durante a definição do processo RUP, foram identificados alguns sintomas comuns recorrentes das falhas de projetos que devem ser levados em consideração para a melhor qualidade do projeto, como: entendimento impreciso das necessidades do usuário; falta de habilidade para lidar com mudanças nos requisitos; problemas de integração em módulos do sistema; dificuldade de manutenção e de evolução; descoberta tardia de sérias falhas do projeto; má qualidade de *software*; performance inaceitável de *software*; falta de rastreabilidade, no sentido de saber que membros da equipe alteraram que parte do sistema; e processo de liberação de versões do *software* não confiável.

5.5 Objetivo do RUP na qualidade de *Software*

Na teoria dos 4 “Quatro P’s” (PRESSMAN, 1995) são definidos o escopo de como se mantém qualidade dentro de um produto de *software* a partir dos seguintes princípios:

- só pessoas (informáticos, gestores e utilizadores) motivadas e comprometidas com o projeto garantem o respectivo sucesso;
- só um processo com técnicas e regras bem definidas permite atingir os objetivos propostos;

- só compreendendo as necessidades reais dos utilizadores se podem produzir um produto de qualidade;
- só com um projeto credível e controlado é possível cumprir prazos e custos propostos.

O principal objetivo do RUP é assegurar uma produção de alta qualidade de *software*, que atenda a necessidade do usuário seguindo prazos e orçamento.

6 QUALIDADE DE *SOFTWARE* COM RUP

Na necessidade de atender seus clientes com qualidade, empresas de *software* se preocupam cada vez mais em utilizar recursos eficazes para construção de produtos de utilização eficiente, mantendo seus negócios viáveis.

Segundo Booch, Rumbaugh e Jacobson (2005, p3), o principal produto é um bom *software* capaz de satisfazer às necessidades de seus usuários e respectivos negócios. Tudo o mais é secundário.

O RUP (*Rational Unified Process*) é um processo de desenvolvimento de *software* que possui um conjunto completo de atividades que define quem faz o que, quando e como.

É usada uma abordagem de orientação a objetos em sua concepção e é projetado e documentado utilizando a notação UML (Linguagem Unificada de Modelagem) para ilustrar os processos em ação.

6.1 Modelagem

Dentro da metodologia do RUP para desenvolvimento de software podemos utilizar a ferramenta de notação UML, para modelar as iterações do sistema criado.

Modelagem é uma parte central de todas as atividades que levam à implantação de um bom software. Construímos modelos para comunicar a estrutura e o comportamento desejados do sistema. Construímos modelos para visualizar e controlar a arquitetura do sistema. Construímos modelos para compreender melhor o sistema que estamos elaborando, muitas vezes expondo oportunidades de simplificação e reaproveitamento. Construímos modelos para gerencias os risos, Booch, Rumbaugh e Jacobson (2005, p4).

6.2 Artefatos produzidos em cada fase

Dentro do processo de desenvolvimento de um produto de software podemos utilizar em todas as fases de construção artefatos com notações em UML. Desta maneira facilita a construção por demonstrar sempre parte de um processo que ao longo de seu desenvolvimento pode ser modificado.

6.2.1 Diagrama de caso de uso

Este diagrama é concebido logo depois que temos o documento de requisitos do sistema a ser criado e tem o objetivo de identificar todos os recursos que os clientes esperam

que o sistema ofereça suporte, mas ele não revela qualquer detalhe sobre implementação desses recursos. (PENDER, TOM/UML, A BÍBLIA, 2004).

No exemplo abaixo temos a imagem de um digrama de caso de uso de um sistema de gerenciamento de *hardware* e *software*.

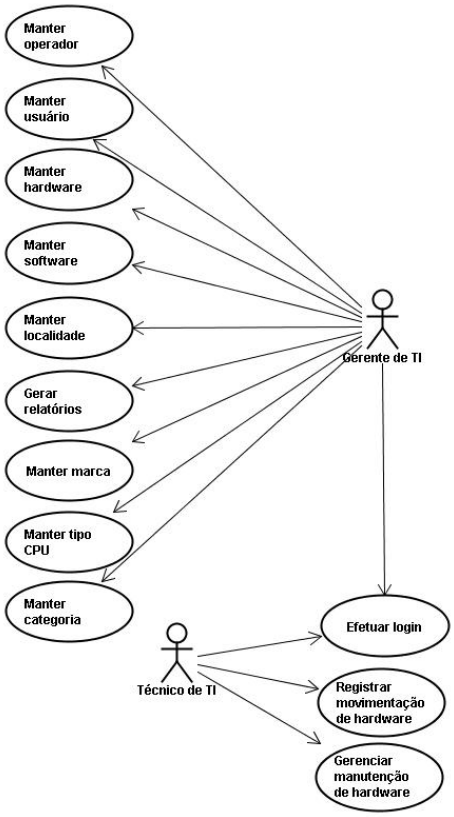

Figura 6-1 - Diagrama de Caso de Uso de um Sistema de Gerenciamento de *Hardware* e *Software*

Fonte: *Autoria Própria (Murilo Andrade) para exemplificação de modelagem de sistemas*

Um caso de uso é uma técnica de modelagem usada para descrever o que um novo sistema deve fazer. Ele é construído através de um processo interativo no qual as discussões entre o cliente e os desenvolvedores do sistema conduzem a uma especificação do sistema das quais todos estão de acordo. Apresentando uma linguagem simples e de fácil compreensão representa de forma prática a interação entre o usuário de um sistema e o próprio sistema.

Suas características principais: Iterativo e Incremental, que corresponde um meio de dividir grandes projetos em projetos menores ele foi elaborado para superar as complexidades determinada pelo modelo cascata, que é utilizado com muito sucesso em projetos pequenos, onde o problema é bem conhecido e controlável.

O sistema é dividido em pequenas partes que são inteiradas. Essa iteração segue o formato sequencial tradicional, a cada interação é identificada as necessidades, analisada,

projetada, implementada e testada, desta forma o sistema será incrementado até que todo o ciclo de desenvolvimento esteja finalizado. Esse modelo iterativo e incremental tem sido bem aceito e utilizado por várias metodologias de desenvolvimento de *software*.

6.2.2 Diagrama de sequencia

Nesta notação em UML temos diagramas comportamentais que são utilizados para visualização, especificação, construção e documentação dos aspectos dinâmicos de um sistema. Considere os aspectos dinâmicos de um sistema como uma representação de suas partes que sofrem alterações (BOOCH, RUMBAUGH, JACOBSON, 2000, p94).

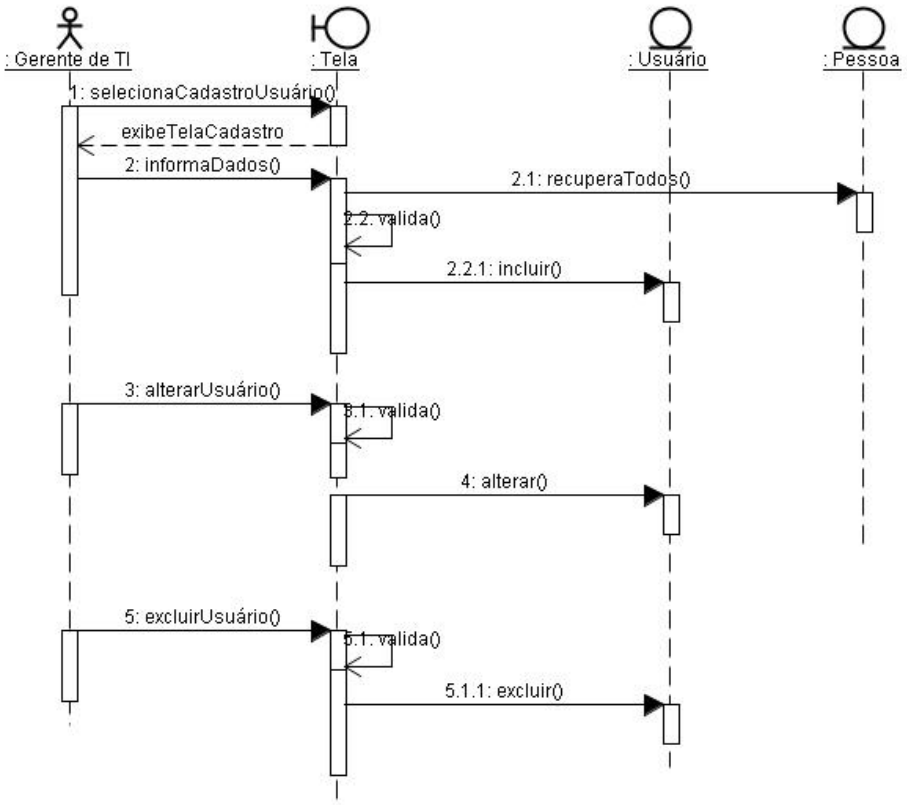

Figura 6-2 – Diagrama de Sequência de Manter Usuário

Fonte: *Autoria Própria (Murilo Andrade) para exemplificação de modelagem de sistemas*

É um modelo de processo moderno derivado também do Processo Unificado de Desenvolvimento de *Software* associado. Normalmente descrito a partir de três perspectivas:

- uma perspectiva dinâmica que mostra as fases ao longo do tempo;
- uma perspectiva estática que mostra atividades de processo;
- uma perspectiva prática que sugere boas práticas (SOMMERVILLE, 2006).

O objetivo do RUP é assegurar uma produção de alta qualidade de *software*, que realiza a necessidade do usuário seguindo prazos e orçamento.

Para que seja atendida a qualidade do *software* o mesmo no seu processo deverá atender os requisitos e padrões, onde o padrão poderá ser o RUP. Cujo os requisitos de *software* são a base a partir da qual a qualidade é medida. A falta de conformidade com os requisitos significa falta de qualidade; e os padrões especificados definem um conjunto de critérios de desenvolvimento que orientam a maneira segundo a qual o *software* passa pelo trabalho de engenharia. Se os critérios não forem seguidos, o resultado quase que seguramente será a falta de qualidade. Esse processo para se alcançar a qualidade está representado na figura abaixo:

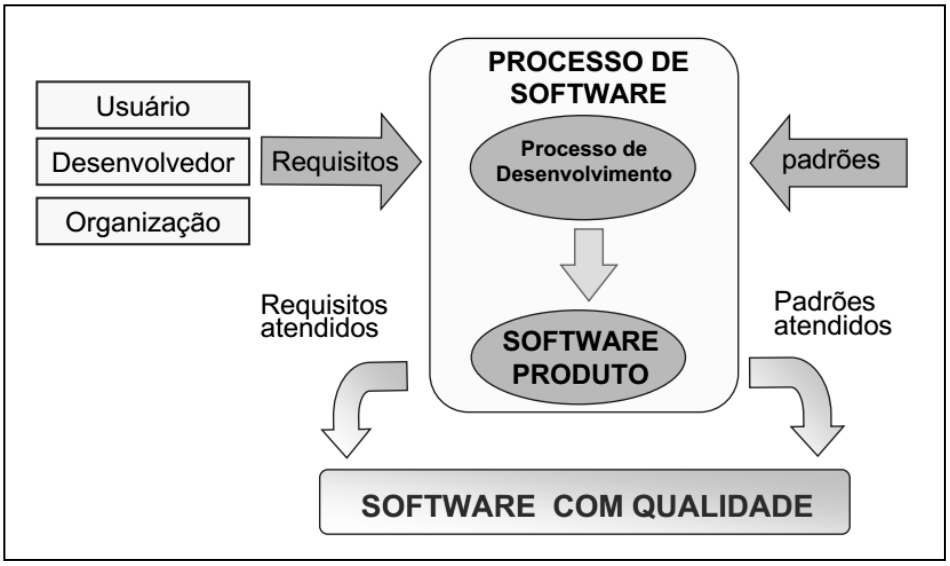

Figura 6-3 - Arquitetura Geral RUP

Fonte: Material dos slides da disciplina de Engenharia de *Software* do professor Ricardo Argenton Ramos da Univasf[3]

Conforme Kruchten (2003), um produto de qualidade deve ter ausência de defeitos e, principalmente, deve atender aos propósitos desejados. Alguma coisa com qualidade deve fazer o que as pessoas querem que ela faça. Se alguma coisa é livre de defeitos, mas não faz o que as pessoas querem que ela faça, essa coisa é um produto desnecessário.

No desenvolvimento de *software*, a qualidade de projeto abrange os requisitos, as especificações e o projeto do sistema. A qualidade de conformidade é assunto concernente, principalmente, à implementação. Se a implementação segue o projeto e o sistema resultante

[3] Disponível em: <http://www.univasf.edu.br> Acesso em 08 de novembro de 2013.

satisfaz os requisitos e metas de desempenho, a qualidade de conformidade é alta (PRESSMAN, 1995).

Robert Glass [GLA98] alega que uma relação mais “intuitiva” é oportuna:

“Satisfação do usuário = produto adequado + máxima qualidade + entrega dentro orçamento e do cronograma.”

A qualidade de *software* não pode ser avaliada de maneira isolada. *Software*s são desenvolvidos pelas organizações através de procedimentos. Um método pobre ou a ausência de uma metodologia pode ser a causa da baixa qualidade. Sendo assim, a avaliação da qualidade está diretamente relacionada com a qualidade de processos e metodologias utilizadas.

Garantia da qualidade é o ponto mais comum de falha nos projetos de *software*, desde isto é frequentemente algo que não se pensa previamente e algumas vezes tratado por equipes diferentes. A RUP ajuda no planejamento do controle da qualidade e cuida da sua construção em todo processo, envolvendo todos os membros da equipe. Nenhuma tarefa é especificamente direcionada para a qualidade; o RUP assume que cada membro da equipe é responsável pela qualidade durante todo o processo. O processo foca na descoberta do nível de qualidade esperado e prove teste nos processo para medir este nível Boente (2003).

Dentro dos processos de metodologia com qualidade garantida temos a ferramenta RUP que satisfaz todos os requisitos para criação de sistemas com robusta documentação. Através da experiência que diversas fábricas traduzem a respeito do RUP, podemos concluir que ainda hoje é uma ferramenta da grande ascensão para desenvolvimento de *softwares* com qualidade absoluta.

7 CONCLUSÃO

Este trabalho teve o objetivo de expor a importância do RUP na qualidade de *Software* em todas as suas fases de desenvolvimento através das normas de Qualidade de Processo, assim como as normas e padrões que avaliam a qualidade do produto já finalizado, através das normas de Qualidade de Produto.

O detalhamento ocorreu na norma de Qualidade de Produto de *Software* ISO/IEC 9126 e 14598, onde suas divisões foram mostradas. Com isso, foi possível perceber que há maneiras efetivas de verificar se um produto de *software* é adequado aos diferentes tipos de usuário e suas respectivas necessidades, pois a norma possibilita, através da utilização de métricas, avaliar diferentes características de um mesmo produto.

Pode-se dizer que o objetivo principal foi atingido, uma vez que se tratava da avaliação da qualidade de *software* baseado nos conceitos do RUP.

As práticas recomendadas pelo RUP foram identificadas em um grande número de projetos de sucesso e sua falta dói verificada em um grande número de projetos fracassados.

8 REFERÊNCIA BIBLIOGRÁFICA

ABNT. NBR ISO/IEC 9126-1 - **Engenharia de *software* – Qualidade de produto** - Parte 1: Modelo de qualidade, Rio de Janeiro: ABNT, 2003.

BARTIÉ, Alexandre. **Garantia da qualidade de *software***: Adquirindo maturidade organizacional. Rio de Janeiro: Campus, 2002.

BROOKSHEAR, J. Glenn. **Ciência da Computação Uma Visão Abrangente**. 5. ed. São Paulo, Bookman, 2000.

CÔRTES, M. L , Thelma C. dos Santos CHIOSSI. **Modelos de Qualidade de Software**. Ed. da Unicamp, 2001 - 152 páginas

FERREIRA, Kelcia; DRUMOND, Elayne. **Normas ISSO para Usabilidade.** Disponível em: <http://www.dcc.ufmg.br/~clarindo/arquivos/disciplinas/eu/material/seminariosalunos/normas-iso-kecia-elayne.pdf>. Acessado em 8 de janeiro de 2010.

KOSCIANSKI, André, *et al*. **Guia para utilização das normas sobre avaliação de qualidade de produto de *software*** - ISO/IEC 9126 e ISO/IEC 14598. Curitiba: ABNT, 1999.

LAUDON , Kenneth C. Jane Price Laudon. **Gerenciamento de Sistemas de Informação**. LTC Editora, 1999.

PRESSMAN, R. S., 1995. **Engenharia de Software**. Makron Books. São Paulo. Brasil.

PRESSMAN, R. S. **Engenharia de software**, Tradução da 5ª Edição. São Paulo: Mc Graw Hill, 2002

PRESSMAN, Roger S. **Engenharia de *Software***. São Paulo: Pearson Makron Books, 2006.

RATIONAL UNIFIED PROCESS. Disponível em: <http://www.wthreex.com/>. Acessado em 01 de dezembro de 2013.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de *software***. 1ª Edição. São Paulo: Prentice Hall, 2001.

WINTER, L. A.; CORDENONZI, W. **Avaliação de Qualidade de *Software* Educacional**.

SOMMERVILLE, Ian. **Engenharia de Software.** 6ª Edição. Addison Wesley, 2004.

SOMMERVILLE, Ian. **Engenharia de *Software***, 8ª. edição, 2006.

MACHADO, Felipe Nery Rodrigues. **Métricas e Qualidade de *Software*.** 1a Edição. Editora Érica, 2011.