UNIVERSIDADE ESTADUAL DA PARAÍBA
CAMPUS VII – GOVERNADOR ANTÔNIO MARIZ
CENTRO DE CIÊNCIAS EXATAS E SOCIAIS APLICADAS
CURSO DE LICENCIATURA EM COMPUTAÇÃO

KÉCIO DA SILVA SANTOS

A IMPORTÂNCIA DA LINGUAGEM DE MODELAGEM UNIFICADA COMO ARTEFATO DA DOCUMENTAÇÃO DE SOFTWARE LIVRE

PATOS – PB
2012

KÉCIO DA SILVA SANTOS

A IMPORTÂNCIA DA LINGUAGEM DE MODELAGEM UNIFICADA COMO ARTEFATO DA DOCUMENTAÇÃO DE SOFTWARE LIVRE

Monografia apresentada ao Curso de Licenciatura em Computação, do Centro de Ciências Exatas e Sociais Aplicadas, da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do título de Licenciado em Computação.

Área de Concentração: Engenharia de Software

Orientadora: Profª. Msc. Ana Carolina Costa de Oliveira

Co-Orientador: Profº. Dr. Elder Eldervitch Carneiro de Oliveira

PATOS – PB
2012

KÉCIO DA SILVA SANTOS

A IMPORTÂNCIA DA LINGUAGEM DE MODELAGEM UNIFICADA COMO ARTEFATO DA DOCUMENTAÇÃO DE SOFTWARE LIVRE

Monografia apresentada ao Curso de Licenciatura em Computação, do Centro de Ciências Exatas e Sociais Aplicadas, da Universidade Estadual da Paraíba, em cumprimento à exigência para obtenção do título de Licenciado em Computação.

Aprovada em ____/____/_____

Ana Carolina Costa de Oliveira

Profª. Msc Ana Carolina Costa de Oliveira / IFPB
Orientadora

Elder Eldervitch Carneiro de Oliveira

Profº. Dr. Elder Eldervitch Carneiro de Oliveira / UEPB
Co-Orientador

Francisca Wilma Cavalcante

Profª. Msc Francisca Wilma Cavalcante / UEPB
Examinadora

Vitor Abílio Sobral Dias Afonso

Profº Esp. Vitor Abílio Sobral Dias Afonso / UEPB
Examinador

Dedico este trabalho aos meus pais Graça e Dalmo, meus exemplos de vida, pelo grande incentivo durante todo esse período, me proporcionando oportunidades que eles nunca tiveram. A minha esposa Ramayana, por todo amor, compreensão e companheirismo durante todo esse tempo junto. Por sempre ter me ouvido, entendido as minhas ausências e estado ao meu lado tanto nos momentos de alegria quanto nos momentos difíceis. Eu te amo.

AGRADECIMENTOS

Dentre todas as pessoas que participaram direta ou indiretamente desta conquista, cito um agradecimento especial.

O maior de todos os agradecimentos, a Deus, que me deu a força necessária para que eu vencesse todos os obstáculos e persistisse nesta caminhada, não me deixando desistir mesmo nos momentos mais difíceis, nada conseguiria se Ele não permitisse e de nada valeria se sem Ele eu vivesse.

A minha família, que sempre me fez acreditar no amor e nos sonhos. Aos meus pais, Graça e Dalmo, pelo constante apoio, carinho e amizade sempre dedicados. A minha irmã Karen, por toda a força e alegria.

A minha esposa Ramayana, pelo incentivo e apoio durante todo esse período e por ter me ajudado com o seu carinho e ter me apoiado com seu entusiasmo para que eu concluísse este trabalho e vencesse todas as dificuldades encontradas.

A minha orientadora e estimada Ana Carolina, por seu profissionalismo, sua paciência e por todo aprendizado que me proporcionou. Por todas as vezes que me compreendeu e que esteve disponível para sanar as minhas dúvidas. Por sempre ter acreditado no meu trabalho e pela sua amizade.

Aos professores da graduação do Campus VII da UEPB, os quais me deram toda base de conhecimento na área de computação, e também pela amizade e colaboração.

Aos "sobreviventes" da minha turma Bruno, Betoven, Clenilson, Ayslane, Raony, Manoel, Ayslânia, Angélica e Raniere, que apesar das dificuldades e aperreios encontradas no curso, sempre estivemos juntos para vencer todos os obstáculos, ficarão na memória os melhores momentos.

As minhas amigas em especial Iskaime e Magaly, que foi uma das melhores coisas que me aconteceu durante o curso, conhecer verdadeiros amigos, a vocês sou muito grato por compartilhar todos os momentos, agradeço muito pelo apoio nas decisões da minha vida. E a todos outros que de alguma forma contribuíram para esta vitória, seja através de torcida, de apoio ou de compreensão.

Kécio Santos

RESUMO

A acelerada difusão do software livre vem se mostrando cada vez mais evidente nos mais diversos âmbitos. O desenvolvimento dessa nova modalidade é considerado inovador comparado aos processos tradicionais. No entanto, a produção de software livre vem acompanhada de algumas questões que precisam de uma melhor atenção, pois esse modelo de desenvolvimento não é realizado seguindo os métodos, técnicas e ferramentas adotadas pela engenharia de software, que buscam qualidade e produtividade nesse processo. Durante o processo de desenvolvimento de software livre, diversos artefatos são produzidos, entre eles, pode-se citar a documentação que tem como finalidade registrar e comunicar todo o funcionamento do sistema, mas por motivos de prazos, custos elevados e outros fatores acabam por comprometer o método de documentação, levando-se a softwares que não apresentam qualidade. Para resolver esse problema, a modalidade de desenvolvimento de software, necessita de uma metodologia que priorize a sua documentação desde as fases iniciais até a própria geração do código. Nesse contexto, este trabalho propõe uma integração entre a UML (Unified Modeling Language) e o método de documentação utilizado no desenvolvimento de Software Livre. Em particular, o trabalho procura mostrar como o uso da modelagem UML na documentação proporciona uma simplificação da complexidade do desenvolvimento de softwares atuais, além de criar softwares mais completos e de qualidade. O resultado desta pesquisa é útil na definição de um processo de documentação mais eficaz para os softwares livres, tendo em vista que apresenta o propósito de auxiliar na criação de novos documentos e dar suporte às atividades de entendimento e manutenção do software.

PALAVRAS-CHAVES: Software Livre. Documentação de Software. UML.

ABSTRACT

The accelerated diffusion of the free software has become more and more evident in the most several contexts. The development of this new modality is considered innovator compared to the traditional process. However, the production of free software comes with some issues that need a better attention because this development model is not accomplished following the methods, techniques and tools adopted by the software engineering, that seek quality and productivity in that process. During the process of free software development, several workmanships are produced, among them, the documentation can be mentioned, that has the purpose registering and communication of the whole system operation, but because of deadlines, high costs and other factors lead to a compromise of the documentation method, causing the low quality of softwares. To solve that problem, the modality of software development needs a methodology that prioritizes its documentation from the initial phases to the own generation of the code. In this context, this study proposes an integration between UML (Unified Modeling Language) and the documentation method used in the development of Free Software. Particularly, the study attempts to show how the use of the modeling UML in the documentation provides a simplification of the complexity development of the current softwares, besides creating more complete and qualified softwares. The result of this research is useful in the definition of a process of more effective documentation for the free softwares, having in mind that model presents the purpose of helping in the creation of new documents and giving support to the understanding activities and maintenance of the software.

KEYWORDS: Free software. Software Documentation. UML.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIAÇÕES

AT&T	American Telephone and Telegraph
CA	Código Aberto
FSF	Free Software Foundation
GNU	*GNU's Not Unix*
OMG	Object Management Group
OMT-2	Object Modeling Technique
OO	Orientação a Objetos.
OOSE	Object-Oriented Software Engineering
OSS	Open Source Software
RFP	Request for Proposals
RS	Rational Software
RTF	Revision Task Force
RUP	Rational Unfied Process
SL	Software Livre
SO	Sistema Operacional
TI	Tecnologia da Informação
UML	Linguagem de Modelagem Unificada

SUMÁRIO

CAPÍTULO 1 – ASPECTOS INTRODUTÓRIOS

Neste primeiro capítulo será apresentado o problema da pesquisa, os principais objetivos traçados e a justificativa do estudo. Ao fim do capítulo são descritos a organização e o conteúdo dos capítulos.

1.1 FORMULAÇÃO DO PROBLEMA

O movimento de Software Livre (SL) tem recebido cada vez mais espaço e relevância nos segmentos da comunidade de software (governo, academia, indústria etc), tanto em esfera nacional quanto mundial. Esse tipo de programa não traz consigo exclusivamente inovações no formato de se desenvolver software, mas ao mesmo tempo proporciona uma nova filosofia, afetando muitos dos atuais princípios da indústria de software (SILVA, 2006).

Embora o seu inegável crescimento, na maior parte das ocasiões, seu desenvolvimento não tem sido concretizado segundo os melhores métodos da Engenharia de Software, não abrangendo nesse panorama, o emprego de processos de softwares bem definidos. Segundo Pressman (2002), a engenharia de software pode ser percebida como a sistematização do processo de desenvolvimento de software, através do bom emprego de padrões técnicos e ferramentas. Seu principal objetivo é alcançar metas, geralmente focadas na qualidade, no custo e no atendimento a prazos.

Atualmente, há um amplo número de projetos de SL em construção e esse número é crescente (SOURCEFORGE, 2006). Em meio aos projetos existentes, alguns têm qualidade estimável. E segundo Bertollo (2006), é largamente notável que a qualidade dos produtos de software resultam da qualidade dos processos de software usados em seu desenvolvimento e manutenção.

De acordo com Falbo (2005), no decorrer do processo de desenvolvimento de software, diversos artefatos são produzidos e modificados a todo momento, evoluindo até que suas finalidades essenciais sejam atendidas. A documentação de código do software por sua vez, pode ser considerada um dos elementos essenciais na produção e manutenção desses sistemas, e é considerado por Forward (2002), um artefato cuja finalidade seja comunicar a informação sobre o sistema de software ao qual ele pertence.

Neste contexto, Nunes, Soares e Falbo (2004) afirmam ser importante o uso de ferramentas para amparar o processo de documentação de software. No meio de inúmeros

artefatos, a Linguagem de Modelagem Unificada (UML) é avaliada por Dantas (2001) como um dos mais importantes artefatos das etapas de análise e projeto, e é analisada como modelos que simulam o problema sendo consertado em um nível de abstração mais superior do que a visão computacional, permitindo um melhor entendimento das inúmeras visões do projeto e servindo como base para as etapas posteriores do processo, como codificação e validação.

Uma documentação de qualidade apresentada por Falbo (2005) propicia uma maior coordenação durante o desenvolvimento de um sistema, facilitando alterações e posteriores manutenções. Além disso, restringe o impacto da perda de membros do grupo, diminui o período de desenvolvimento de etapas posteriores, encurta o tempo de manutenção e colabora para diminuição de erros, somando assim, a qualidade do processo e do produto gerado. Dessa forma, a construção da documentação é tão importante quanto a criação do software em si (SANCHES, 2001).

Sousa et al., (2004), afirmam que a ausência de documentação em softwares livres pode representar um risco para os projetos de software, especialmente na etapa de manutenção. Nenhuma pessoa pode garantir que a informação, que está na mente dos desenvolvedores experientes, continuará claro na equipe quando esta for diluída. Segundo Freeman e Munro (1992), a carência de documentação ou a documentação incorreta danifica a interpretação do sistema, principalmente na atividade de análise de código.

Dessa forma, torna-se necessário que no desenvolvimento de softwares livres, a documentação, tenha como elemento integrante a construção de modelos consistentes, segundo uma notação estabelecida e padronizada – a UML – como um fator relevante para o sucesso do desenvolvimento de software (DANTAS, 2001).

Baseada nessas premissas, a questão que a pesquisa se propõe a responder é **por que documentar os softwares livres utilizando a Linguagem de Modelagem Unificada - UML?**

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Apresentar a modelagem UML como componente da documentação no processo de desenvolvimento de software livre.

1.2.2 Objetivos Específicos

1. Mostrar a modelagem UML como forma de padronizar a documentação de software livre, tornando-a mais simples de ser modificada;
2. Descrever a UML como meio de abstração da complexidade do software e simplificação da realidade, possibilitando uma maior qualidade na documentação;
3. Apontar as vantagens e desvantagens de utilizar a modelagem UML na documentação de softwares.

1.3 JUSTIFICATIVA

A documentação é uma atividade fundamental na produção de software (NUNES et al., 2004). Dentro do processo de desenvolvimento a documentação não faz parte de uma etapa definida, mas acontece durante toda a sua existência, em paralelo com outras etapas de desenvolvimento e manutenção (LACERDA, 2005).

Sanches (2001) afirma que é através dela que a evolução do software é registrada para que sejam criadas as bases necessárias para as etapas posteriores do processo de software, incluindo treinamento, utilização e manutenção de software.

Contudo, diferentes fatores, como prazos apertados, elevados custos, indefinição e dificuldade na manipulação de documentos, podem comprometer o método de documentação, induzindo a documentos inacabados, desatualizados e incoerentes (SANCHES, 2001).

Na opinião de Bertollo (2006) a falta de documentação se deve também à grande complexidade dos sistemas atuais e à expectativa de que eles tenham alta qualidade nas funções projetadas e que sejam desenvolvidos sem a necessidade de alocação de mais recursos.

Neste sentido, a ausência da documentação na visão de Silva (2004) tem sido a principal franqueza de muitas organizações de Tecnologia da Informação (TI), que a tem deixado de lado, seja por falta de uma política organizacional ou inexistência de uma ferramenta específica para essa atividade, o que faz com que desenvolvedores optem por deixar a documentação em segundo plano.

A pouca atenção dada à documentação de softwares livres é vista na percepção de Christoph (2004), como consequência do próprio caráter informal de um projeto deste tipo, e o resultado é que as únicas fontes de documentação disponíveis são normalmente o código fonte e os arquivos contendo mensagens trocadas entre os participantes do projeto.

A importância de documentações sobre funcionamento, composição e estruturação de um software se torna cada vez mais relevante a medida em que a complexidade e as funcionalidades do software se tornam maiores. No contexto de um software livre, esta documentação se torna ainda mais relevante devido ao grande número de desenvolvedores (SANTOS, 2011). De fato, um software pode se tornar descontinuo quando não recebe manutenção de seus criadores e, sem uma completa documentação, torna-se mais simples desenvolver um novo software do que prover manutenção de software.

Uma investigação recente feita por Reis (2003) com diversos projetos de software livre, comprova que existe uma preocupação por parte dos projetos com a documentação, mas essa documentação é desenvolvida sem formalidades, centralizada mais no próprio código fonte e menos em documentação externas. Além do mais, a documentação externa tem vista no usuário final e não o desenvolvedor. Algumas informações significativas da pesquisa são:

- apenas 30% dos projetos analisados responderam que desenvolvem e conservam documentação formal para os desenvolvedores;
- apenas 24% dos projetos se aproveitam de algum tipo de padrão para a codificação do software;
- a maioria dos projetos (78%) apresenta algum tipo de documentação para o usuário final;
- apenas 55% destes projetos asseguram que uma parte expressiva de sua documentação é modernizada frequentemente.

Para Christoph (2004), esta falta de documentação pode gerar em quaisquer projetos, um indesejado obstáculo para entrada de novos participantes na evolução do software. Desta forma, se o obstáculo de entrada for maior do que o empenho do desenvolvedor, este não colaborará com o projeto, o que é uma dificuldade para projetos de software livre que dependam exclusivamente de colaboradores voluntários para sua continuidade.

Nunes et al., (2004) afirma que é importante analisar que não basta apenas produzir uma documentação completa, é necessário também que a documentação formada acompanhe atenciosamente as manutenções e alterações nos sistemas, para que seja conservada a integridade e veracidade da informação contida na documentação.

Dados estatísticos apontados na pesquisa de Coelho (2009) mostram que o tempo gasto em manutenção de sistemas varia de 47% a 62%, e esse tempo é despendido na tentativa de compreender os aplicativos, e que fazer leitura do código fonte custa 3,5 vezes mais do que

ler a documentação, além do mais, os dados mostram ainda que 53% das falhas são achados em trechos apresentados como corretos.

Além disso, é de conhecimento geral que um código bem constituído torna o trabalho mais simplificado, e caso este código não se encontre satisfatoriamente bem documentado ou escrito, as evoluções serão realizadas com dificuldade (LEHMAN, 1996). Por isso a documentação de software torna-se importante por aumentar a produtividade do projeto e código, além de fornecer mecanismo para a geração de artefatos. (LACERDA, 2005)

Garofalo e Campos (2010) consideram que as documentações do sistema compreendem todos os documentos e artefatos que delineiam o seu desenvolvimento, da especificação de requisitos ao plano de teste final para aceitação pelo cliente. Dentre estes documentos destaca-se a UML, que segundo Almeida e Darolt (2001) representa uma maneira de padronizar a documentação, com vistas a que qualquer sistema possa ser modelado perfeitamente, com coerência, com fácil comunicação com outras aplicações, simples de ser modificado e compreensível.

Ferreira, Moura e Carvalho (2006) consideram que modelar software é a capacidade de simplificá-lo em uma linguagem que admita facilitar e reter informação de sua cobertura. Assim, a UML apresenta uma denotação bem definida, alusivo aos símbolos empregados no processo de documentação, o que admite a qualquer desenvolvedor ser apto de interpretar tais símbolos sem ambiguidade. Esses símbolos são usados para produzir diagramas, que são capazes de ser considerados como o cerne da UML (CRUZ, 2006).

Apesar dessa adoção, pode-se expor que ela incidiu de cima para baixo, uma vez que um bom número de desenvolvedores e diretores de projeto ainda resiste a UML, ou melhor: resistem à inserção do processo de documentação no seu processo de desenvolvimento (MOURÃO, 2007). De fato para o autor, ainda é muito natural observar equipes cuja documentação dos sistemas se abrevia aos artefatos de baixo nível tais como códigos fonte, scripts de banco e às vezes diagramas de entidade e relacionamento, acessíveis unicamente aos participantes da equipe técnica.

Para Lacerda (2005) a UML é uma linguagem designada para visualização, especificação, construção e documentação de artefatos de software, admitindo a visualização da aplicação sob várias perspectivas. A omissão da modelagem torna esses processos extremamente onerosos e o custo do aprendizado da equipe leva, prematuramente, à utilização de técnicas de elaboração ou reconstrução do sistema (FERREIRA et al., 2006).

Um dos grandes benefícios de se usar modelos formais na percepção de Fowler (2004) é a simplificação do diálogo entre os participantes da equipe de produção e com o próprio

usuário e/ou analista de negócios, uma vez que inúmeros dos diagramas previstos na UML são compreensíveis por pessoas com pouca ou nenhuma qualificação técnica, a partir da informação básica dos elementos de alto nível.

Os artefatos que compõem a documentação produzida no desenvolvimento de softwares, como por exemplo, a UML, são considerados na visão de Falbo (2005) como entradas para as atividades de garantia da qualidade, quando os mesmos são averiguados quanto à aderência em relação aos padrões de documentação e validados em relação aos seus propósitos que se propõem a atender.

Ferreira et al., (2006) comenta que a medida que o processo de documentação se materializa o custo completo das manutenções irá inevitavelmente diminuir, tornando o processo de manutenção mais viável para as organizações. Sendo assim, utilizar a UML de maneira ciente e madura significa dar um passo importante em direção à criação de softwares mais completos, com mais atributos e com menos esforços (PENDER, 2004).

Resumindo todos os motivos apontados para a modelagem da documentação de software, a UML na visão de Ferreira (2006) torna-se ferramenta importante para aumentar a eficiência da comunicação entre os desenvolvedores da aplicação, proporcionando um padrão para abstração da complexidade ou simplificação da realidade, e possibilitando uma maior qualidade na documentação dos softwares.

Por fim, esta pesquisa foi concentrada em uma metodologia que busca conjugar conceitos de diversas áreas para a constituição da metodologia deste trabalho.

1.4 ASPECTOS METODOLÓGICOS

Inicialmente será apresentada a definição da metodologia, juntamente com as delimitações do método. Posteriormente, a formulação de hipóteses e para finalizar a estrutura usada neste trabalho.

1.4.1 Definição da Metodologia

A natureza da pesquisa apresentada neste trabalho, pode-se afirmar que é de natureza básica, pois busca gerar novos conhecimentos úteis para a ciência.

Do ponto de vista da abordagem do problema, a pesquisa é considerada qualitativa, pois há uma predominância de análises mais dissertativas. Foram avaliados e discutidos

artigos científicos, relatórios técnicos, livros e trabalhos acadêmicos que tratam sobre Software Livre, Documentação de software e Modelagem UML.

Este trabalho teve início com uma revisão bibliográfica sobre os principais temas que o compõem, com a meta de alcançar os objetivos que foram propostos, e através do estudo foram definidas as especificações do trabalho.

A pesquisa trata-se, portanto, de um estudo exploratório com características que combinam, ao mesmo tempo, um levantamento teórico e uma análise simplificada dos métodos que não utilizam a modelagem com os que a usam na documentação.

1.4.2 Delimitações do Método

O estudo realizado durante este trabalho voltou-se para uma das etapas do processo de desenvolvimento de Software Livre que é a documentação do código, enfatizando-se os artefatos que são produzidos no decorrer deste processo e as ferramentas usadas para amparar a documentação.

Neste contexto, o estudo do processo de produção da documentação será centrado na utilização do artefato de modelagem UML, como ferramenta que busca reduzir a abstração dos códigos de softwares, permitindo um maior entendimento do projeto. Isso garante uma documentação de qualidade, que facilita futuras alterações e manutenções.

1.4.3 Formulação de Hipóteses

A hipótese a ser examinada nesta pesquisa é que a documentação de código torna-se um diferencial para os softwares que a possuem no seu processo de desenvolvimento, apesar de não ser dada tanta ênfase para os softwares de natureza livre. O estudo também analisa a inclusão da ferramenta de modelagem UML na produção da documentação de SL para proporcionar a criação de softwares mais completos e de qualidade.

1.5 ESTRUTURA DO TRABALHO

Este trabalho está dividido em cinco capítulos. Nesta primeira etapa, foi apresentado o problema da pesquisa, os principais objetivos traçados, as justificativas do estudo e ao fim do capítulo foram descritos a metodologia utilizada e a organização deste documento.

A revisão da literatura será dividida será divida em 03 capítulos, englobando os aspectos relacionados a Softwares Livres (SL), Documentação de Software e a Linguagem de Modelagem Unificada (UML).

No segundo capítulo apresentam-se questões introdutórias acerca do Software Livre, seu histórico, definição desses softwares, a comunidade de SL, Open Source e os processos usados no seu desenvolvimento (definição e suas características) que serão abordados no decorrer deste trabalho.

O terceiro capítulo considera o processo de produção da documentação dos softwares, fazendo-se uma abordagem mais especifica para a documentação de SL. Serão expostos os conceitos, tipos, ferramentas e usos da documentação, como também os diversos artefatos que são produzidos no decorrer do processo, dando ênfase para a modelagem UML.

O quarto capítulo analisa a utilização da modelagem UML durante uma das fases do desenvolvimento de software, a produção da documentação. Será apresentado a UML como uma linguagem padronizada para a construção de modelos e a importância da consistência destes modelos para o sucesso de desenvolvimento. Além disso, mostra o porquê de utilizar a modelagem UML na documentação de código, como suas vantagens e desvantagens.

No último capítulo constaram as considerações finais mais importantes deste trabalho, como a apresentação dos principais objetivos atingidos e suas soluções, suas principais contribuições e limitações, aspectos positivos e negativos, assim como perspectivas de trabalhos futuros.

CAPÍTULO 2 – SOFTWARE LIVRE

Neste capítulo apresenta-se questões introdutórias acerca do Software Livre, seu histórico, definição desses softwares, a comunidade de SL, Open Source e os processos usados no seu desenvolvimento (definição e suas características) que serão abordados no decorrer deste trabalho.

2.1 DEFINIÇÃO DE SOFTWARE LIVRE

O termo Software Livre - SL (Free Software) faz referência a softwares que são fornecidos aos seus usuários com a liberdade de usar, copiar, estudar, modificar e distribuir (com ou sem modificações) sem que, para isso, os usuários tenham que solicitar permissão ao criador do programa, em particular, a possibilidade de alterações implica que o código fonte esteja disponível (HEXSEL, 2002).

Segundo a Free Software Foundation – FSF (2012) o SL é uma questão de liberdade, não de valor. Mais precisamente, se refere a quatro liberdades para seus usuários, que segue:

0. Liberdade de executar o programa, para qualquer propósito;
1. Liberdade de estudar como o mesmo funciona e adaptá-lo para as suas necessidades;
2. Liberdade de redistribuir cópias de modo a ajudar ao próximo;
3. Liberdade de modificar o programa e liberar estas modificações, de modo que toda a comunidade se beneficie.

As liberdades 1 e 3 aludem na necessidade de se ter acessível o código-fonte, apontado como a principal particularidade dos software livres.

Na década de 1980, a ideia de SL se difundiu, com a criação da Free Software Foundation (FSF), por Richard Stallman. O movimento de SL surgiu de uma disputa aos mercados proprietários mais influentes da indústria de software, tornar-se visível todo um apelo político, institucional e emocional (SOFTEX, 2005).

Diversos produtos de software livres oferecem determinadas características vantajosas e típicas, muitas vezes, resultantes do próprio processo de desenvolvimento desse grupo de produtos de software, Nakagawa (2002), compreendendo estabilidade, portabilidade para uma multiplicidade de plataformas, acesso ao código-fonte, apoio aos usuários por parte dos desenvolvedores, baixo custo e evolução contínua, com versões mais novas do software sendo

produzidas e lançadas com maior constância e defeitos sendo retificados em um tempo menor do que em softwares pago (DAVIS et al., 2000).

O SL por possuir um estilo diferenciado de desenvolvimento em relação aos demais softwares, exigi uma nova metodologia de produção, para que se garanta a qualidade desses produtos. Na sequência será mostrado como é dado o funcionamento das comunidades de desenvolvimento desses softwares.

2.2 A COMUNIDADE DE SOFTWARE LIVRE

O processo de produção de software livre tem se contornando uma importante área de estudo e pesquisa da Engenharia de Software, inúmeras comunidades vem alcançando sucesso mundial através deste modelo de desenvolvimento (LIMA, 2005).

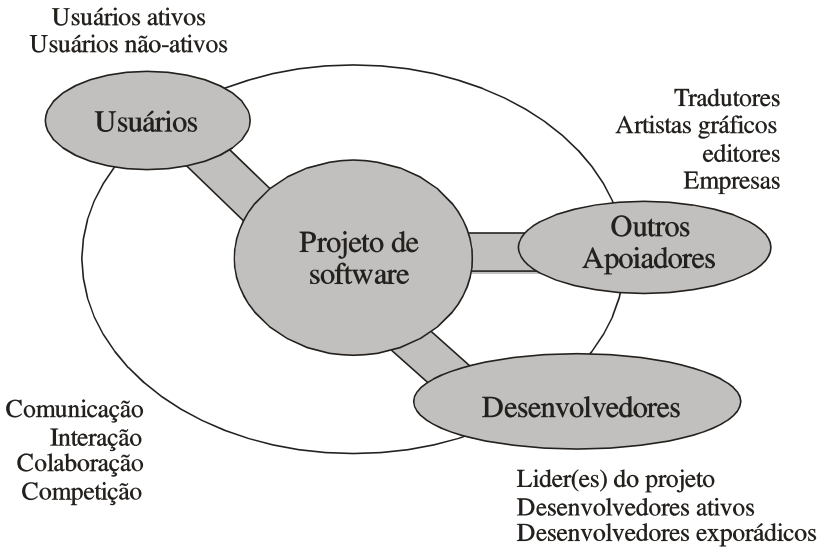
A comunidade de software livre na visão de Christofh (2004) é formada por integrantes altamente colaborativos que fazendo parte em um mesmo projeto, não se conhecem pessoalmente já que na maior parte dos casos se localizam separados por amplas distâncias geográficas, comunicando-se apenas através de listas de e-mail, espaços virtuais para o armazenamento de código e outras ferramentas disponíveis na Internet.

Para a maioria dos casos, isso é verdade, mas cabe lembrar que existem exceções, não é raro que membros de um projeto desta categoria façam todos parte de uma mesma empresa, que pode vir até dar início a um projeto como comercial e mais tarde modificar-se em um software livre, pode-se mencionar como exemplo, softwares famosos como o Mozilla, o compilador Java Jikes da IBM e o Java Development Kit da Sun (GODFREY, 2000).

Salles Filho et al., (2005) apontam que estas comunidades podem ser, por exemplo, grupos de usuários, grupos de desenvolvedores, grupos mistos, grupos de debates técnicos, grupos de organização e articulação, grupos que utilizam/desenvolvem um software em comum. A Figura 01 ilustra essas participações, buscando representar a inclusão dos diversos autores em um projeto. Este último tipo de grupo é o que mais usualmente se define como comunidade: aquelas que se formam em torno de um software ou de projeto de desenvolvimento de software.

Outra particularidade de destaque destas comunidades é o fato de que a maioria das pessoas envolvidas em projetos de software livre acaba por trabalhar como voluntário e não ganha retorno financeiro, observa-se que essa característica também não se aplica a todos os casos, já que existem diversos projetos desta modalidade que são feitos ou até favorecidos por empresas (REIS, 2003).

Figura 01 – Representação esquemática das comunidades que se relacionam com SL



FONTE: Filho et al., (2005)

Para Lima (2005) as respostas alcançadas por membros destes projetos compreendem o prestígio dentro da comunidade de software, o progresso do software em questão (já que comumente o membro usufrui deste software) e o fato de estar colaborando para a ampliação do software livre no mercado atual.

Tendo em vista que os membros das comunidades de desenvolvimento de SL estão realizando um trabalho voluntário, isso traz certas implicações que podem se tornar diferenças decisivas entre um projeto de software livre e um projeto habitual. Algumas destas diferenças são apontadas na pesquisa de Godfrey (2000) e estão relacionadas a:

- **Tempo de entrega:** Como o trabalho realizado é voluntário, o membro utiliza apenas seu tempo disponível para trabalhar no projeto, não tendo compromisso com datas de entrega.
- **Motivação dos desenvolvedores:** Como comentado no tópico anterior o desenvolvido é voluntário, o projeto é algo que o desenvolvedor tem empenho em fazer, mesmo que este empenho revelar-se somente por uma determinada área do projeto e não pelo todo, sendo suficiente para que a motivação deste membro seja alta.
- **Tempo de lançamento das novas versões:** Por não se ter compromisso com prazos, o projeto só será lançado quando seus criadores entenderem que ele está o suficiente maduro, pois em projetos tradicionais que dependem de prazos apertados, pode vir a gerar o lançamento prematuro de um software.

- **Testes, manutenção, planejamento e documentação:** Estas etapas para evolução em um projeto de software livre nem sempre são consideradas tão atraentes quando programar, resultando na perca de qualidade nessas etapas. A qualidade do código em um projeto de software livre é mantida na maioria das vezes por massivos testes paralelos (grupo de usuários examinando e testando o software e citando bugs encontrados), ao invés de testes sistemáticos.

A duração das comunidades de SL na concepção de Salles Filho et al., (2005) está relacionada diretamente com o interesse das pessoas com relação a estas comunidades e ao projeto desenvolvido, outro motivo que pode levar comunidades de SL a deixar de operar é a falta de liderança definida, nesses casos um novo líder com desejo de dar continuidade ao projeto, deve ocupar este espaço e manter a comunidade em atividade.

Para Lima (2005) o propósito das comunidades de SL pode ser determinado pelo projeto que a comunidade desenvolve. Desta forma, a necessidade da realização do projeto é dividida pelos membros que formam a comunidade, sendo o público alvo do projeto e o interesse pelos projetos fatores que podem determinar o sucesso da comunidade, no qual se mostra claro a preocupação em expor qual o seu propósito, que problemas se propõem a resolver e quem elas objetivam atender, tornando claros e acessíveis os propósitos aos participantes e aos interessados em participar da comunidade.

Na seção seguinte os conceitos de Software Livre e Open Source serão apresentados de modo sucinto, abordando-se a diferenças e os contextos que essas terminologias são utilizados.

2.3 SOFTWARE LIVRE E OPEN SOURCE

Em ambos os conceitos, software livre e código aberto (open source), as mesmas premissas são respeitadas, ou seja, são mantidas as liberdades de copiar, distribuir, estudar, modificar e melhorar o software. Porém estas premissas, em certos casos, são mais restritivas em software livre comparadas a Código Aberto (CA) (LIMA, 2005).

O termo CA é utilizado na maioria das vezes por conta do caráter ambíguo usado no termo software livre, notadamente quando está colocado na língua inglesa, free software, pois a expressão free tanto pode ser entendida no contexto de omissão de custo de aquisição quanto no contexto de liberdade, ou seja, podem ser oferecidas análises diferentes a software livre de acordo com o ambiente em questão (FSF, 2012).

Para Johnson (2001) Software Livre é empregado em maior parte dentro de um contexto ideológico, enquanto código aberto é um termo mais usado com finalidade comercial e de desenvolvimento, a Free Software Foundation – FSF, fundação que promove a utilização e a produção de SL vem em defesa do SL como um direito, ressaltando as obrigações técnicas associadas à distribuição de software e conhecimento. Código Aberto é frequentemente usado para delinear possibilidades de negócios com software livre, destacando aspectos do processo de desenvolvimento e da organização social. Nesse contexto será exposto o termo Open Source de forma a entender o porquê de sua criação e as suas premissas.

2.3.1 Open Source

O termo Open Source é comumente utilizado para descrever software do mesmo grupo que Software Livre, em contextos bastante semelhantes. Por que existe esta variedade de nomes?

A expressão Open Source foi criada em 1998 com a finalidade de desenfatizar o teor filosófico relacionado à liberdade que o termo 'Software Livre' possui (COSTA; SANTOS, 2010). Existia uma motivação para dissolver barreiras de preconceito ligadas à defesa deste princípio, o que aprimoraria a chance de persuadir empresas a aceitar este tipo de software e sua metodologia de desenvolvimento, na prática, uma licença avaliada Open Source (e não de software livre) tende a ser mais permissiva em relação à geração de produtos originados não-livres (SILVA, 2006).

De certa forma, *'open source'* realça os aspectos do processo e da organização social, enquanto *'software* livre' ressalta os aspectos de livre redistribuição e troca de conhecimento. Há certa predisposição de certos grupos, como a FSF, em analisar o termo open source rapidamente menos correto ou digno que free software (REIS, 2003).

Nesta monografia, buscando evitar discussões que não sejam estritamente relacionadas ao processo de utilização da modelagem na documentação dos projetos de software livre, optou-se por usar exclusivamente o termo software livre.

2.3.2 Histórico do Software Livre

Software Livre é um conceito antigo, apesar de não ter sido utilizado esse termo específico até a década de 80, a maioria dos softwares existente nas décadas de 60 e 70 foram

produzidos com uma abordagem colaborativa e aberta em inúmeras empresas e instituições, o Unix original é um exemplo desta tendência (KON; SABINO, 2009).

No decorrer dessa época surgiu a "cultura hacker", visto que os códigos dos softwares eram distribuídos abertamente entre os mais habilidosos e consagrados programadores, especialmente após o aparecimento da primeira rede transcontinental de alta velocidade entre os computadores, a ARPANET (DORNELAS, 2004).

Logo depois da criação da rede de computadores, vários dos empenhos cooperativos na produção de softwares tiveram foco na criação de um sistema operacional que atuasse em diversas plataformas de computadores, o maior sucesso desse empenho foi o nascimento do sistema operacional Unix que, inicialmente, era transferido dentre as instituições gratuitamente ou por uma pequena taxa de instalação (LERNER; TIROLE, 2002). Mas, com o aceleramento da difusão do Unix em 1980, a empresa que o criou, American Telephone and Telegraph (AT&T) passou a exigir os direitos autorais.

Com a determinação da AT&T em não distribuir o sistema operacional Unix com o código aberto, inúmeros esforços foram concretizados para formalizar a cooperação na produção de futuros softwares que pudessem suprir tal sistema operacional (KON; SABINO, 2009). A década de 80 ocasionou-se uma alteração importante: a modificação gradual para licenças restritivas, que não permitiam redistribuição ou alteração. O software fornecido, que até então era visto como uma combinação de código-fonte com código executável passou a significar apenas o executável (DIPOLD, 2005).

Dando prosseguimento será apresentado os primórdios do surgimento do SL, tendo em vista que a criação do SL só foi possível por intermédio do sistema operacional Unix.

2.3.2.1 Unix

A investida inicial para a história do software livre foi dada em 1969, quando Ken Thompson, pesquisador do Bell Labs e um grupo de programadores da AT&T, criou a primeira versão do Unix, um Sistema Operacional (SO) multitarefa, onde vários programas eram executados simultaneamente, e multiusuário, que admitia que vários usuários utilizassem o computador ao mesmo tempo, por meio de terminais remotos (DIPOLD, 2005). Este sistema era usado pelos grandes computadores que havia na década de 70 em universidades e grandes empresas, os mainframes.

O sistema operacional Unix era espalhado gratuitamente para as universidades e centros de pesquisa, com seu código-fonte acessível. A sigla OSS (Open Source Software) é a

que indica essa modalidade de programa, cuja composição pode ser alterada por qualquer usuário com conhecimentos em informática, diferentemente dos sistemas operacionais mais usados ultimamente, como o Windows. A partir daí foram aparecendo novas versões do Unix, igualmente abertas e compartilhadas pelo meio acadêmico (COSTA; SANTOS, 2010).

Com a determinação em exigir os direitos autorais pela utilização do Unix, será abordado na seção subsequente os esforços realizados para suprir tão necessidade, que foi a criação de um SO denominado GNU e da Free Software foundation.

2.3.2.2 GNU e a Free Software Foundation

No ano de 1985, Richard Stallman fundou a Free Software Foundation (FSF), com objetivo de desenvolvimento de um SO totalmente livre. O sistema operacional tinha a finalidade de ser compatível com Unix, e foi denominado de GNU (uma sigla recursiva que significa *GNU's Not Unix*), para garantir sua liberdade, Stallman também criou uma licença para este software, a GNU GPL, a licença de software livre mais utilizada (REIS, 2003). Para o autor a tarefa de desenvolver o sistema operacional abrangia não apenas a criação de um núcleo de sistema operacional, mas também a criação de uma coleção de bibliotecas e aplicativos que admitissem ao sistema compatibilidade com o Unix original.

Costa e Santos (2010) afirmam que o processo de desenvolvimento desta época não era documentado na literatura, mas é certo dizer que já existia uma comunidade de desenvolvedores interessados nos produtos da FSF e que colaborava com código-fonte e pacotes para aperfeiçoar as lacunas no sistema operacional GNU.

Logo após a criação do GNU e da Free Software Foundation, diversos softwares foram sendo desenvolvidos utilizando as mais variadas metodologias de desenvolvimento, na seção seguinte serão apresentados os principais métodos usados na produção de SL atualmente.

2.4 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE LIVRE

Essa seção visa dar uma visão sobre o processo de desenvolvimento de softwares livres, sua definição e algumas características da produção de SL.

2.4.1 Definição

Um processo de software pode ser determinado como a união coerente de atividades, políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos indispensáveis para idealizar, desenvolver, distribuir e conservar um produto de software (FUGGETTA, 2000). Para que o processo ocorra de maneira eficaz necessita, nitidamente, considerar as relações entre as atividades, os artefatos produzidos no desenvolvimento, as ferramentas e os procedimentos necessários e a habilidade, o treinamento e a motivação dos recursos humanos envolvidos (FALBO, 2005).

A seleção de um modelo de processo é o ponto de partida para a determinação do processo de desenvolvimento de software, e se tratando de Software Livre a preferência deve ser fundamentada em um procedimento que priorize essa nova metodologia de desenvolvimento, que agrega à sua disposição ferramentas de comunicação e trabalho colaborativo (REIS, 2003).

Tendo em vista que o processo de desenvolvimento de SL é o marco inicial para a qualidade do produto desenvolvido, será mostrado algumas características de como é realizado esse processo utilizando metodologias de desenvolvimento voltadas para SL.

2.4.2 Características do desenvolvimento de Software Livre

Pelo seu caráter cooperativo, propiciado principalmente pela distribuição do código fonte dos programas, Silva (2006) aponta que o desenvolvimento de Software Livre é feito por pessoas que colaboram geograficamente dispersos, em sua maioria, ao redor do mundo, esse novo formato de produção demanda uma abordagem de desenvolvimento diferenciada.

É evidente a diferença entre trabalhar com grupos próximos, bem orientados, com regras e procedimentos instituídos por organizações, quando no desenvolvimento de software comercial, e realizar o desenvolvimento de software livre, em que regras e normas são bem flexíveis e podem ser adaptadas quando uma particularidade manifestar-se interessante e necessária (FREITAS, 2009).

A metodologia utilizada na produção de software livre pode ser compreendida como um processo de trabalho virtual, por não haver estatutos ou regras pré-definidas, diferentemente das metodologias seguidas em empresas e instituições que produzem software (FREITAS, 2009). Também não há, em software livre, compromisso qualquer de confidencialidade, tendo em vista, que um projeto de Software Livre é uma organização composta por um conjunto de pessoas que utiliza e desenvolve um único software livre, colaborando para uma base comum de código-fonte e conhecimento (BRITO et al., 2004).

Christoph (2004) mostra que não há um processo de desenvolvimento de software livre avaliado como "padrão", diferentes desenvolvedores e comunidades utilizam técnicas, métodos e ferramentas diferentes. Para o autor, apesar de que estas diferenças existam, ainda assim é possível descrever nos diferentes projetos inúmeras características comuns. Apresentam-se aqui estas características, advertindo que estas podem vir a passar por modificações significativas em alguns projetos mais específicos.

O modelo de desenvolvimento de software tradicionalmente usado para produzir software proprietário ou acadêmico é expresso no artigo The Cathedral and The Bazaar de Modelo Catedral (RAYMOND, 1999). Sua principal característica é sua forma centrada e controlada de desenvolver o software, havendo a necessidade de uma pessoa (ou grupo de pessoas) que centralize o processo de desenvolvimento.

Quando se reflete em software livre, o Modelo Catedral não é apropriado para seu desenvolvimento, principalmente pela dispersão geográfica dos inúmeros desenvolvedores. Devido a esse fato, Raymond (1999) sugeriu a adoção de um modelo chamado Modelo Bazar para a produção de SL, contendo características peculiares, tais como: disponibilização de código fonte; envolvimento de múltiplos desenvolvedores, muitas vezes, voluntários; processos colaborativos dos mesmos, que estão dispersos geograficamente; prazo pequeno de desenvolvimento; e liberdade para os desenvolvedores elegerem o trabalho que almejam realizar.

O Modelo Bazar tem sido utilizado com sucesso em vários projetos, tais como os projetos do Linux e do Apache. Nota-se, contudo, que para se conseguir sucesso na sua aplicação, é imprescindível, ainda, ter (NAKAGAWA, 2002): metas bem definidas do que será produzido; uma motivação simples de ser entendida; um bom líder que alimente os desenvolvedores com motivação e o objetivos em mente; uma comunidade de membros que trabalhe com ânimo e de forma descentralizada; e uma tecnologia que permita a comunicação de forma eficiente. Dessa forma, o sucesso do Modelo Bazar não depende unicamente da disponibilização do código fonte, mais necessariamente da organização e coordenação de todo o processo de forma apropriada (SILVA, 2006).

Reis (2003) buscou levantar e quantificar aspectos essenciais do processo utilizado nos projetos de software livre, dentre os conclusões a que ele chegou em sua pesquisa, podem-se destacar:

C1. Na maior parte dos projetos de SL, os membros trabalham de maneira descentralizada, envolvendo muitas vezes, desenvolvedores que não se conhecem pessoalmente.

C2. Na maioria dos projetos de SL, os desenvolvedores são também usuários do software em desenvolvimento e colaboram efetivamente para definir sua funcionalidade.

C3. Há na maior parte dos projetos de SL, um sistema de liderança, sendo as formas mais utilizadas: líder com delegação informal e líder com delegação formal;

C4. Existe um uso vasto de ferramentas para viabilizar a comunicação entre os participantes da equipe geograficamente dispersa, com ênfase para ferramentas de controle de versão e listas de correio eletrônico, além de sistemas de acompanhamento de alteração/defeitos e ferramentas de comunicação em tempo real.

C5. Grande parte dos projetos de SL possui equipe reduzida.

C6. Existe pouca documentação formal de projeto e é provável que grande parte do trabalho de descrever o projeto fique a cargo do próprio código-fonte.

Com tudo isso, as características apresentadas expõem o grande diferencial dessa nova concepção de desenvolvimento de software, que é o acesso ao código-fonte, proporcionando-se uma reflexão sobre a importância da documentação do código desses projetos de SL, que permite que o software seja desenvolvido com mais qualidade.

2.5 CONCLUSÕES DO CAPÍTULO

Nesta seção foi apresentado uma visão geral do SL, os diferentes contextos de desenvolvimento, além das características particulares usadas na produção desses softwares.

O SL traz, de fato, variáveis novas para a indústria de software. Embora não se trate de uma ruptura tecnológica, traz uma nova forma de desenvolver e licenciar software, quebrando alguns modelos estruturais de apropriabilidade nesta indústria.

Neste contexto são usadas muitas formas de organização para o desenvolvimento de software, algumas mais abertas e horizontais (o que a literatura chama de modelo Bazar), outras mais verticais e hierárquicas (também chamado de modelo Catedral). Entre um e outro modelo, há hoje uma gama relativamente extensa de situações.

Assim, as formas variadas de organização de SL atendem a interesses diferenciados, e essas metodologias devem priorizar todas as fases do desenvolvimento, principalmente artefatos que valorizem a documentação nesse processo. No próximo capítulo será apresentado com mais detalhes o processo de se documentar os SL, como também ferramentas que auxiliam essa metodologia.

CAPÍTULO 3 – DOCUMENTAÇÃO DE SOFTWARE LIVRE

Neste capítulo faz-se uma descrição do processo de produção da documentação dos softwares, fazendo-se uma abordagem mais especifica para a documentação de SL. Serão expostos os conceitos, tipos, ferramentas e usos da documentação, como também os diversos artefatos que são produzidos no decorrer do processo, dando ênfase para a modelagem UML.

3.1 DOCUMENTAÇÃO DE SOFTWARE

Desde cedo, a apreensão com o processo de se documentar um sistema de forma concisa e unificada passou a ser apreciada principalmente a partir da década de sessenta, um ponto de destaque neste processo foi à documentação por Ivar Jacobson de um sistema de telecomunicação através do emprego de camadas de blocos (FERREIRA, 2006). Apesar de ser perceptível que desde décadas passadas já existam sistemas com uma documentação adequada, a qual admitiria que modificações pudessem ser feitas com o mínimo consumo de tempo e recursos financeiros, até hoje se pode com facilidade localizar softwares essenciais às organizações que não dispõem de uma adequada documentação (DANTAS, 2001).

Ambler (2001) sugere dois motivos básicos para documentar, um serve para auxiliar a comunicação no decorrer do projeto e outro para auxiliar a compreensão nas atividades de manutenção, a documentação é indispensável quando é preciso estabelecer uma comunicação com uma equipe externa de trabalho, o que é o caso no desenvolvimento de SL, pois os seus desenvolvedores estão localizados em regiões geograficamente distintas. Para o autor ela serve como mecanismo de suporte a comunicação quando é combinada com reuniões, teleconferência, correio eletrônico e ferramentas colaborativas.

A documentação de software tem um resultado expressivo no entendimento do programa. Segundo Falbo (2005), mais de 50% do trabalho de evolução do software é destinado ao entendimento do programa e o trabalho mais difícil da manutenção de software é o entendimento da estrutura do sistema.

Uma documentação de qualidade é extremamente necessária para o sucesso do desenvolvimento e, uma vez que implica inteiramente na qualidade do processo e do produto originado, precisa ser construída e acessada de forma exclusiva e padronizada, necessitando de procedimentos para seu controle (FERREIRA, 2006).

Fundamentado nestas premissas, a questão é: qual é a documentação mínima que sirva para o propósito de auxiliar o entendimento do sistema na fase de desenvolvimento e manutenção? Portanto, o trabalho mostrar o uso de uma ferramenta de software de maior importância para a atividade de obter o entendimento do sistema na fase de produção dos artefatos de documentação, a modelagem UML. O resultado desta investigação será favorável no emprego desse artefato no processo de desenvolvimento de software cuja finalidade da documentação seja o de dar suporte às atividades de manutenção e entendimento do software.

3.1.1 Definição

Primeiramente, necessita-se conceituar a expressão documentação, ou seja, o que é documentação?

Para Garofalo e Campos (2010) a documentação é um conjunto de documentos que permitem o acesso a informações sobre um software, seja para seu uso, para informação do seu funcionamento ou para a cooperação com o seu desenvolvimento. É o registro que admite a autonomia de novos usuários (sejam leigos ou experientes, programadores, desenvolvedores etc.) na utilização do programa.

Nas palavras de Matte (2008) a documentação pode ser analisada como documentos que torna um código aberto acessível aos interessados: são documentos que esclarecem como funciona, como foi produzido, como pode ser utilizado e o que foi realizado em cada atualização. Documentação, portanto, é a ferramenta que proporciona o acesso ao conhecimento.

Documentação de software pode ser determinada como um componente com o propósito de transmitir a informação sobre o sistema de software ao qual ele faça parte (FORWARD, 2002). No entanto, é necessário apontar a diferença entre modelos, documentos, código fonte e documentação.

Segundo Ambler (2001), do ponto de vista da modelagem ágil, um documento é qualquer artefato externo ao código fonte cuja finalidade seja transmitir conhecimento de um modo constante. Modelo é uma abstração que delineia um ou mais aspectos de um problema ou de uma solução potencial para resolver um problema. Determinados modelos podem virar documentos, ou compreendidos como parte deles, a exemplo da modelagem UML que permite a simplificação da abstração do sistema e o seu entendimento. Código fonte é uma sequência de instruções, abrangendo os comentários que apresentam estas instruções, para um

sistema de computador. O termo documentação contém documentos e comentários de código fonte.

A documentação de software na percepção de Sousa et al. (2004) tem um grande valor para a Engenharia de Software. Várias pesquisas estão sendo realizadas para diminuir as dificuldades que envolvem a documentação de software: processos de documentação onerosos e custosos, documentação em fartura e sem finalidade, documentação desatualizada e de baixa qualidade, problema de acesso, entre outros.

Alguns pesquisadores sugeriram a utilização de hipertexto (texto em formato digital) para promover o acesso à documentação. Freeman e Munro (1992) propuseram combinar documentação e código fonte de um modo simples e eficiente, empregando ferramenta de hipertexto. Ouchi (1985) definiu uma estrutura de dados úteis para dar subsídio a um sistema de documentação voltado para manutenção de software e advertiu sobre a importância de técnicas padronizadas de documentação.

O HCi Journal (2001) destacou que antes do início de cada projeto, se faz indispensável um planejamento da documentação a ser empregada, pois todo desenvolvimento tem particularidades peculiares. Lethbridge et al., (2003) sugeriram a necessidade de se empenhar na produção de um simples e poderoso formato e ferramenta de documentação, ao invés de forçar os engenheiros de software atingir um trabalho dispendioso e ineficaz. Cioch; Palazzolo e Lohrer (1996) propõem uma abordagem de documentação que considere o tipo de informação necessária para cada estágio de aprendizagem (recém-chegado, aprendiz, internos, experientes).

Nota-se que os estudos sobre documentação de software tendem a resolver problemas de falta de atualização, dificuldade de acesso, falta de qualidade, desorganização, documentação desnecessária e indicam para soluções simples, com o apoio de ferramentas e que sirvam a um propósito. Na próxima seção será apresentado como a documentação pode utilizada de diferentes formas, inclusive quando se pensa na liberdade dos SL.

3.1.2 Relação de documentação e Liberdade

Refletindo-se em software livre, é admissível afirmar que uma documentação adequada tende a fortalecer a liberdade (GAROFALO; CAMPOS, 2010). Para os autores, a filosofia por trás dos softwares livres pressupõe o seu tratamento não como objeto de posse, não como produto, mas sim como ideia, qualquer coisa que possa ser compartilhado sem que isso lhe desagregue valor. Conforme Silveira (2004, p. 11), "o software livre possui um autor

ou vários autores, mas não possui donos". Assim, o software livre se orienta por princípios éticos, antes mesmo de seus pressupostos técnicos e tecnológicos.

Diante do exposto, passa a existir em contextos mais amplos e habituais do desenvolvimento e utilização de softwares, determinadas perguntas visivelmente óbvia, mas que nem sempre são explicitadas. Como copiar, distribuir, modificar ou publicar cópias de um programa sem a preocupação de deixar um registro para o usuário que receberá essa cópia? Como um usuário principiante poderá usar e tirar proveito dessa cópia sem a informação necessária? Como conquistar novos usuários, num mercado no qual predominam grandes empresas, se não é proporcionado meios de conhecimento aos novos usuários? Como esperar que os usuários, já tão habituados com o amparo técnico pago de softwares proprietários, resolvam seus próprios problemas sem fontes de consulta e solução de dúvidas eficientes?

Para aproximar-se mais dos ideais de liberdade, defendidos pelo software livre, Garofalo e Campos (2010) relatam que é preciso uma documentação que ampare tanto o usuário inexperiente no primeiro contato com os programas, como também os que estão vinculados ao seu processo de produção. Torna-se necessário que o usuário mais experiente, que contribui com o desenvolvimento de programas, deixe registrada, para os futuros interessados, a solução de casuais problemas, para que não se repercutam a cada novo usuário.

É preciso que todo o processo de alteração e construção de um software, independentemente da dimensão da contribuição ou de sua complexidade, seja registrado para oferecer transparência a alguma pessoa interessada em contribuir (AMBLER, 2001). Para tanto, é fundamental meditar sobre os objetivos do desenvolvimento de um software livre e de sua concepção, a partir dos quatro princípios fundamentais mencionados anteriormente e conhecidos por todos os que, de alguma forma, embarcam no contato com o uso e/ou o desenvolvimento desses softwares.

Nunes et al., (2004) afirmam que com o auxílio da documentação, é provável garantir os princípios mais elementares do software livre e a continuação de sua colaboratividade. Coloca-se, então, outra questão: considerando que é importante documentar, como produzi-la para que se alcancem seus objetivos? Será feita uma reflexão sobre essa questão na sessão 3.4, que mostrará o processo de produção da documentação de software. Ainda para responder a pergunta, torna-se indispensável saber qual o tipo de documentação que será utilizado para alcançar tais objetivos, esses tipos serão apresentados na próxima seção.

3.2 TIPOS DE DOCUMENTAÇÃO

Segundo Michelazzo (2006), a documentação pode ser dividida em dois amplos grupos, conforme a Quadro 1.

A documentação técnica é considerada mais simples, pois apresenta o trabalho do desenvolvedor, enquanto que a documentação para o usuário é a mais exigente e demanda habilidades específicas para a redação de manuais, inserção de screenshots, desenhos e outros elementos gráficos.

Quadro 01 – Tipos de documentação de software

DOCUMENTAÇÃO TÉCNICA	DOCUMENTAÇÃO DE USO
• Voltada ao desenvolvedor, programador e etc. • Compreende dicionários e modelos de dados, fluxogramas de processos e regras de negócios, dicionário de funções e comentários de código.	• Voltada para o usuário final e o administrador do sistema • Formada por apostilas, tutoriais, ou manuais que apresentam como o manual deve ser usado, o que esperar dele e como receber as informações que se deseja.

FONTE: Coelho (2009)

Para Nunes (2005) no desenvolvimento de sistemas de software, deve haver, dentre outros, documentação para delinear processos (plano de projeto, plano de testes etc.), descrever a composição do sistema a ser arquitetado (modelos de análise e projeto, código fonte etc), descrever o produto originado (manual do usuário, ajuda online, tutoriais etc.) e descrever a organização em si (capacidades dos recursos humanos, recursos de hardware e software disponíveis, documentos de problemas e soluções etc.).

Qualquer software necessita ter uma quantidade admissível de documentação, seja documentos de trabalho ou manuais de usuário criados profissionalmente, em projetos de SL a documentação vem deixando a desejar no critério de qualidade e padronização.

O registro da documentação precisa ser feito levando-se em conta os diferentes tipos de usuário, é fundamental observar que os usuários podem ser assim classificados: usuários finais – que utilizam o software para ajuda-lo em alguma tarefa e não estão preocupados em detalhes técnicos e administrativos; e os administradores do sistema – responsável pela administração do software, a exemplo, operadores, gerentes de rede, etc. (SOUSA et al., 2004). Além disso, para os autores alguns critérios são importantes na composição da documentação do usuário, como a descrição funcional do sistema, manual de introdução, manual de referência que apresenta as facilidades do sistema e seu uso, documento de instalação, manual de administrador do sistema e ajudas on-line.

Para Michelazzo (2006) a documentação do sistema (técnica) apresenta o software sendo implementado, desde a especificação de requisitos até o plano de teste, e o mais importante é que sua estrutura seja produzida induzindo a especificações mais detalhadas do

sistema, pertencem a esta documentação os documentos de requisitos, descrição da arquitetura do sistema, documentos de validação e guias de manutenção.

Quando se fala em documentação técnica faz-se referência a todo e qualquer artefato que seja criado no decorrer do processo de desenvolvimento que tenha um caráter informativo ou de comunicação entre as partes envolvidas no projeto (COELHO, 2009). Especificações, definições de caso de uso, documentação de fontes, diagramas e modelos são exemplos de documentação técnica.

Sendo assim, a documentação para técnicos contém determinados termos comuns entre as comunidades que estão ligadas inteiramente com essa linguagem em seu trabalho (desenvolvedores, designers, programadores, etc.). Esses vocabulários não podem ser utilizados na documentação direcionada aos usuários, uma vez que o seu público-alvo possuem disparidades, podendo conter leigos ou iniciantes (GAROFALO; CAMPOS, 2010).

Neste trabalho optou-se em enfatizar o uso da documentação técnica, mais especificamente mostrar o uso de modelos da linguagem UML na documentação, como uma forma de minimizar os problemas de manutenção e consequentemente de evolução de um software. Seguindo esse contexto e dando sequência ao trabalho na próxima seção serão mostradas algumas das ferramentas que são utilizadas para produzir essas documentações, como também seus usos nos diferentes contextos de software.

3.3 FERRAMENTAS PARA DOCUMENTAÇÃO E SEUS USOS EM SOFTWARE

O objetivo desta seção é apresentar de maneira sucinta as ferramentas que são usadas no processo de documentação de softwares e como é feita a utilização dessa documentação em diversas situações.

3.3.1 Ferramentas utilizadas na documentação

Existe um provérbio árabe que diz: “Conheces um trabalhador pelas tuas ferramentas” (BUBACK, 2007). Para o autor a existência de softwares cada vez mais complexos, prazos reduzidos e a necessidade de despesas menores, em função de um comércio cada vez mais competitivo, a produção de software tornou-se uma trabalho árdua. Listar requisitos, produzir documentação, seguir cronograma, gerenciar riscos, codificar, testar, implantar são algumas das atividades presentes em qualquer ciclo de desenvolvimento de software que se tornaram praticamente impossíveis de ser alcançadas sem o uso de ferramentas apropriadas.

Para Buback (2007) a utilização de ferramentas que dê suporte a geração dos documentos pode tornar a atividade de documentação mais acelerada, além de otimizar a atualização quando tais documentos são modificados. Ainda, a ideia de criar modelos de documento facilita a padronização dos documentos de uma organização.

Com a separação da documentação em duas grandes áreas mostradas anteriormente, torna-se importante conhecer algumas ferramentas que auxiliam e/ou facilitam aqueles que têm pela frente o trabalho de gerar documentação de sistemas.

Segundo Michelazzo (2006) destacam-se algumas ferramentas:

- **Documentação de código:** feita basicamente de duas formas – comentários inclusos no próprio código e geração de documentação online. O desenvolvedor precisa ter a consciência de que não será o único a "colocar a manusear o sistema" e por isso deve fazer o comentário dos códigos de maneira bastante clara, pois não é a quantidade de linhas de comentários que garante a eficácia da documentação, mas a omissão de comentários poderá ocasionar gastos inúteis de horas de trabalho para a busca de erros e provavelmente provocará estresse em todos que irão lidar com o software.
- **Modelos de dados:** refletem de forma gráfica (e lógica) a base de dados de um sistema, seus relacionamentos, entidades, chaves e tudo aquilo que se refere aos dados em si. É, portanto, analisado como peça fundamental para o desenvolvimento de um sistema e por isso são pensados e criados antes do início do desenvolvimento. Podem ser criados por meio de engenharia reversa ou ainda baseando-se nas necessidades do aplicativo que está sendo envolvido. As ferramentas mais conhecidas para a modelagem dentro do mundo livre são DBDesigner, MySQL Workbench e PGDesigner, as quais possuem funcionalidades diferentes e dispõem de versões tanto para Linux quanto para Windows.
- **Dicionários de dados:** arquivo ou documento que determina a organização básica dos dados do banco. Nele são transmitidas informações como as tabelas, os campos, suas definições, tipos e descrições das variáveis que compõem a documentação.
- **Fluxogramas:** mostram graficamente a sequência lógica das informações de um processo ou sistema, usando diversos elementos de geometrias que indicam uma das partes do processo. Visualmente conseguem passar a lógica de todo

um sistema desde os níveis mais altos de processamento até menores partes, admitindo, assim, uma visão geral do que verdadeiramente necessita ser feito dentro de um sistema.

- **Modelagem UML:** proporcionam uma padronização de modelagem orientada a objetos de forma que qualquer sistema, seja qual for o tipo, possa ser modelado corretamente, com consistência, fácil de se comunicar com outras aplicações, simples de ser atualizado e compreensível. (ALMEIDA; DAROLT, 2001). É importante distinguir entre um modelo UML e um diagrama (ou conjunto de diagramas) de UML, o último é uma representação gráfica da informação do primeiro, mas o primeiro pode existir independentemente.

Diante todas as ferramentas apresentadas, pode-se notar que a documentação de software possui um grande arsenal de ferramentas que a auxilia na sua composição, garantindo o que é fundamental na produção de softwares que é a qualidade. Com isso, na próxima seção será apresentado como essa documentação pode ser utilizada em diferentes contextos.

3.3.2 Usos da documentação de software

Historicamente, documentos vêm constituindo uma das ferramentas chave para registro e disseminação do conhecimento humano como um todo de maneira geral. Para inúmeras campos existentes (negócios, governo e meio científico, por exemplo), a documentação escrita ainda é o meio mais significativo de descrição e comunicação do conhecimento humano (ERIKSSON, 2007).

No contexto de projetos de software, um ponto de destaque é a constante procura e troca de conhecimentos entre os colaboradores especialistas envolvidos para que as atividades sejam concretizadas de forma adequada. Neste cenário, a documentação exerce também um papel importante no decorrer o ciclo de vida dos projetos, conservando não só a comunicação, mas também o entendimento efetivo do projeto (FORWARD; LETHBRIDGE, 2002).

Para Ambler (2001) a documentação de software é utilizada para mediar o diálogo entre os participantes de um grupo de desenvolvimento, como também serve de conhecimento para informar aos usuários como usar e conduzir o sistema e ainda informações para as pessoas responsáveis de realizar a sua manutenção.

Para tanto, a documentação desse processo deve descrever e registrar todo o procedimento de desenvolvimento e manutenção do software produzido. Sendo assim, o processo de produção da documentação deve ser feito seguindo alguns métodos que serão apresentadas na próxima seção.

3.4 PROCESSO DE DESENVOLVIMENTO DA DOCUMENTAÇÃO

Documentar é informar, porém a comunicação sem formalidades, essencialmente baseada em palavras, não a torna satisfatória, além da facilidade de acontecer distorções no entendimento, expressão popularmente conhecida como telefone sem fio, ela não é capaz de resistir ao tempo e não é prática quando a número de pessoas envolvidas no projeto é muito elevado ou quando o mesmo tem rotatividade alta. Por isto, um processo formalizado de documentação se torna necessário (BUBACK, 2007).

Deste modo, a documentação adquire um papel indispensável no processo de desenvolvimento de software, visto que a mesma está presente em todas as fases do ciclo de vida do software (PRESSMAN, 2002). Como a demanda por software aumenta ininterruptamente, especialmente no que diz respeito a problemas que exigem soluções mais complicadas, inúmeras ocasiões o trabalho de se documentar torna-se deveras dispendioso, sendo frequentemente posto de lado em detrimento a outras atividades, tal como a codificação. Entretanto, sem uma documentação sólida e delineada, é praticamente impossível garantir a manutenibilidade (BUBACK, 2007).

As práticas de desenvolvimento constituem a espinha dorsal da produção de software e, de modo geral, são concretizadas conforme uma ordem constituída no planejamento. As atividades de gerência e de controle da qualidade são, na maioria das vezes, ditas atividades de apoio, pois não se encontram ligadas inteiramente à construção do produto final: o software a ser entregue para o cliente, incluindo toda a documentação necessária (FALBO, 2005).

Neste contexto, o enfoque da documentação tem papel fundamental, no qual o prazo de entrega do projeto pode mudar de acordo com o número e a qualidade dos artefatos de software a serem entregues (SOUSA et al., 2004). Um gerenciamento adequado dos artefatos criados na documentação durante a produção de um software torna-se um fator de extrema vigilância e importância para que se chegue a um produto final de qualidade, sem desperdiçar recursos pessoais e financeiros (BRUEGGE et al., 2006).

Segundo Carvalho e Amaral (2010) para ter-se uma documentação completa e bem determinada é necessário garantir que durante o processo de software seja utilizado ferramentas que auxiliarão na tarefa de gerenciar, documentar e armazenar tal documentação de todas as alterações acontecidas pelos mais diversos tipos de artefatos de software. Para tanto, a sugestão do trabalho em utilizar a modelagem UML como artefato traz consigo inúmeros benefícios para a documentação de SL. Esse artefato de modelagem será visto com mais detalhes nas seções seguintes que mostrará como essa ferramenta se adequa na documentação de software.

3.4.1 Artefatos desenvolvidos no processo

Conseguir qualidade nos processos e produtos de documentação incluso na engenharia de software não é uma tarefa trivial, são múltiplos fatores que atrapalham atingir os objetivos de qualidade (FALBO, 2005). No entanto, o autor afirma que grandes volumes de recursos são gastos no desenvolvimento de um sistema, mas em muitos casos ocorre uma grande frustração por parte das pessoas responsáveis pela manutenção e dos clientes diante da forma final apresentada pelo software encomendado, pois o mesmo não oferece uma documentação que minimize o entendimento do software. Segundo Rocha (2001) inúmeros desses problemas são originados da falta de cuidado para o trabalho de decidir e produzir a documentação do sistema.

Como citado anteriormente um aspecto eficaz para a qualidade de um sistema de software é a sua documentação. Artefatos de software são elementos fundamentais em um processo de desenvolvimento, eles são produzidos e aproveitados por atividades do processo e, assim fazem parte de todo o ciclo de vida de desenvolvimento de software (FALBO, 2005). Os projetos de desenvolvimento de software contemporâneos têm progressivamente aperfeiçoado o tamanho e complexidade, sendo cada vez mais notável sua realização por equipes trabalhando ao mesmo tempo e localizadas geograficamente distantes, tornando a documentação e acompanhamento das modificações um trabalho delicado e de fundamental importância para o sucesso do projeto.

Ferramentas provendo facilidades de documentação permitem agilizar o desenvolvimento de software, realizando algumas tarefas automaticamente, e permitem que modificações sejam mais facilmente realizadas, simplificando a tarefa de manutenção. (PETERS; PEDRYCZ, 2001). Entretanto, é difícil encontrar ferramentas nas quais a documentação possa ser feita de forma integrada, customizada e consistente.

Atualmente, a grande maioria das abordagens para solucionar esse tipo de problema é focada em código fonte. Contudo, os sistemas mais complexos demandam um esforço adicional para a sua compreensão, tornando necessário o uso de artefatos que descrevem aspectos não representados pelo código-fonte, agregando informação em um nível de abstração mais elevado (PRESSMAN, 2002).

Neste contexto, usar a modelagem UML como artefato que compõe a documentação dos softwares livre pode ser uma forma de reduzir os obstáculos encontrados na produção da documentação, tendo em vista que essa ferramenta além de proporciona uma abstração da complexidade do sistema, pode ser aplicada em diferentes metodologias de desenvolvimento pela flexibilidade de manuseio.

Através da utilização da UML, os desenvolvedores e os clientes terão maior comunicação e aproveitamento dos modelos produzidos. Para Dantas (2001) a ferramenta tem recebido grande importância quando se trata de documentação, pois atualmente existem inúmeras ferramentas livres de ajuda à criação dos modelos gerados a partir da UML, o que admite até mesmo ao usuário modificar a documentação produzida pela ferramenta de acordo com suas necessidades, pode-se citar algumas ferramentas livres como Dia, Umbrello, ArgoUML e etc. Como o objetivo desse trabalho é apresentar o uso da UML na documentação de SL, na próxima seção é feita uma breve introdução de como esse artefato pode ser empregado na documentação.

3.4.2 Documentação com a UML

A preparação de uma documentação requer uma maior reflexão sobre as prováveis soluções para o software sendo produzido. Neste caso, a equipe de desenvolvimento se vê forçada a endereçar, por exemplo, o modo como as diversas partes do software serão associadas e os erros que provavelmente venham a incidir, antes que qualquer linha de código seja composta, o que possivelmente acarretará uma melhor manutenibilidade deste software (NUNES et al,. 2004).

Para que o software desenvolvido apresente esses preceitos, a ferramenta UML pode ser utilizada para modelar várias fases do sistema, desde os primeiros contatos até a geração do código, e é aproveitada em qualquer tipo de sistemas em termos de diagramas de orientação a objeto, na maioria das vezes essa ferramenta é mais usada na modelagem de softwares usando o conceito de orientação a objetos, mas também pode ser aplicada em outros

sistemas, de engenharia em geral, que pode também ajudar na organização de processos (ALMEIDA; DAROLT, 2001).

Para os autores, o padrão UML é o mais usado hoje em dia, pois com ele há uma documentação interativa e automática dos processos de modelagem, sendo também aceitável a geração, pela ferramenta, de códigos a partir da mesma modelagem.

Os fundadores da UML buscaram desenvolver uma linguagem unificada padrão que pudesse ser de fácil compreensão a todos. Preocuparam-se em deixá-la aberta aos desenvolvedores, onde os mesmos poderiam definir seu próprio método de trabalho. Com isso, cada projeto de desenvolvimento de software pode adequar à ferramenta com as suas realidades e necessidades (DANTAS, 2001).

Se tratando de SL, onde a maioria dos projetos não possui uma especificação bem delimitada e formal, espera-se que o colaborador não só codifique, como também documente o software seguindo um determinado padrão. Padrão esse que pode ser definido com a utilização da modelagem UML para a produção da documentação desses softwares, o que proporcionará pelo menos uma documentação mínima, para, assim, facilitar a evolução do projeto e a entrada de novos colaboradores.

3.5 CONCLUSÕES DO CAPÍTULO

A documentação de software tem grande importância para o processo de desenvolvimento, principalmente por registrar decisões e requisitos que são fundamentais para o projeto. Neste contexto, foi mostrado no decorrer deste capítulo o processo de documentação de forma clara, desde seus conceitos iniciais, até a identificação de ferramentas que podem auxiliar o processo de produção desses artefatos de software de maior importância para a atividade de obter o entendimento do sistema.

O uso de uma ferramenta que apoie a geração dos documentos pode tornar a atividade de documentação mais rápida, além de facilitar a atualização quando tais documentos são alterados. Com isso, integrar a produção da documentação com a ferramenta UML pode viabilizar a criação de modelos de documento que facilita a padronização dos documentos de um grupo de desenvolvimento, permitindo assim a agilidade na produção e na manutenção da documentação.

As diferentes formas de usar e aproveitar os recursos oferecidos pela UML será apresentado no próximo capítulo, que mostrará também os diferentes contextos que pode ser empregada essa ferramenta que criar documentos mais completos e de mais qualidade.

CAPÍTULO 4 – MODELAGEM UML

Este capítulo descreve uma análise do uso da modelagem UML durante uma das fases do desenvolvimento de software, a produção da documentação. Será apresentado a UML como uma linguagem padronizada para a construção de modelos. Inicialmente, é mostrada uma contextualização dessa linguagem, juntamente com os subconjuntos que a compõe. Posteriormente, são apresentadas de forma mais clara como a UML pode ser usada na documentação de Software Livre, mostrando seus benefícios, vantagens e desvantagens.

4.1 A LINGUAGEM UML

A Linguagem de Modelagem Unificada ou Unified Modeling Language (UML) é uma linguagem visual usada para modelar sistemas computacionais através do padrão de Orientação a Objetos (OO). Essa linguagem tornou-se, no decorrer dos últimos anos, a linguagem padrão de modelagem de software seguida internacionalmente pela indústria de Engenharia de Software (BOOCH, 2000). Neste contexto, esta seção aborda os conceitos iniciais da UML, uma breve introdução e seu histórico, aplicações da UML e os padrões utilizados pela linguagem.

4.1.1 Introdução

A grande dificuldade do desenvolvimento de novos sistemas empregando a orientação a objetos nas etapas de análise de requisitos, análise de sistemas e design é que não se tem uma notação padronizada e verdadeiramente eficaz que envolva qualquer tipo de aplicação que se deseje (SILVA; VIDEIRA, 2001). Para os autores toda simbologia que existe tem suas próprias terminologias, conceitos e gráficos, derivando em um grande conflito, principalmente para aqueles que almejam usar a orientação a objetos não só compreendendo para que lado dirige a seta de um relacionamento, mas sabendo criar modelos de qualidade para ajudá-los a estabelecer e conservar sistemas cada vez mais eficazes.

Vargas (2012) afirma que a modelagem de software é a atividade de edificar modelos que esclareçam as características ou o desempenho de um software ou de um sistema de software. Na constituição do software os modelos podem ser utilizados na assimilação das características e funcionalidades que o software precisará prover (análise de requisitos), e no

planejamento de sua construção. Comumente a modelagem de software emprega algum tipo de notação gráfica e são apoiados pelo uso de ferramentas.

A modelagem de software normalmente dar a entender a construção de modelos gráficos que representam os artefatos dos componentes de software usados e os seus inter-relacionamentos (SILVA, 2007). Um modo comum de modelagem de programas orientados a objeto é através da linguagem unificada UML.

Quando a UML foi disseminada, vários desenvolvedores da área da orientação a objetos ficaram entusiasmados já que essa padronização sugerida pela UML era o tipo de força que eles sempre almejaram (GUEDES, 2009). A UML é muito mais que a padronização de uma notação. É também o desenvolvimento de novos conceitos não normalmente utilizados. Por isso e muitas outras razões, a boa compreensão da UML não é apenas instruir-se na simbologia e na sua definição, mas também significa aprender a modelar orientado a objetos no estado da arte (VARGAS, 2012). Na próxima seção será apresentado como se deu a origem dessa linguagem que padroniza os modelos utilizados na documentação.

4.1.2 Histórico

A UML nasceu da união de três técnicas de modelagem: o método de Booch'93 de Grady Booch, o método OMT-2 (Object Modeling Technique) de James Rumbaugh e o método OOSE (Object-Oriented Software Engineering) de Ivar Jacobson. Estes eram, até meados da década de 90, os três métodos de modelagem orientada a objetos mais conhecidos entre os profissionais da área de desenvolvimento de software (GUEDES, 2009). A união dessas metodologias contou com o amplo apoio da Rational Software (RS), que estimulou e financiou a união dos três métodos.

Todos eles possuíam pontos fortes em determinado aspecto. O método OOSE tinha foco em casos de uso (use cases), OMT-2 se sobressai na fase de análise de sistemas de informação e Booch'93 era mais forte na fase de projeto. O reconhecimento desses métodos foi, sobretudo, devido ao fato de não terem arriscado estender os métodos já existentes. Seus métodos já concorriam de maneira independente, então ficaria mais produtivo permanecer de forma conjunta (SAMPAIO, 2007).

O empenho inicial do projeto deu inicio com a união do método de Booch com o método OMT de Rumbaugh, o que derivou no lançamento do Método Unificado no final de 1995. Logo em seguida, Jacobson juntou-se a Booch e Rumbaugh na Rational Software e seu método OOSE começou também a ser agrupado à nova metodologia. O trabalho de Booch,

Jacobson e Rumbaugh, conhecidos popularmente como "Os Três Amigos", resultou no lançamento, em 1996, da primeira versão da UML propriamente dita (FOWLER, 2004).

Logo após a versão inicial ter sido lançada, diferentes empresas influentes na área de modelagem e desenvolvimento de software passaram a colaborar com o projeto, provendo sugestões para aperfeiçoar e expandir a linguagem. Em 1997 a UML foi aceitada e adotada pelo Grupo de Gerenciamento de Objetos (OMG - Object Management Group) como uma linguagem padrão de modelagem (ALMEIDA; DAROLT, 2001).

A OMG lançou uma RFP (Request for Proposals) para que mais empresas pudessem cooperar com a evolução da UML, resultando à versão 1.1. Depois de obter esta versão, a OMG passou a adotá-la como padrão e a se responsabilizar (através da RTF – Revision Task Force) pelas revisões. Essas revisões são de certo modo, aconselhadas a não gerar uma grande modificação no escopo original, observando-se as diferenças entre as versões existentes, notar-se que de uma para a outra não existi grande impacto, o que tornou mais simples sua dispersão pelo mundo (VARGAS, 2012). Com inúmeras versões da UML sendo produzidas, a seção seguinte apresenta como essas versões são aceitas no processo de produção de software.

4.1.3 Aceitação da linguagem

Para definir a UML como modelagem padrão, os desenvolvedores e a Rational Software compreenderam que a linguagem teria que encontrar-se disponível para todos, consequentemente, a linguagem não deveria ser restrita e sim aberta a todos. Com isso, várias empresas se sentiram livres para usá-las com seus métodos adequados e para criarem ferramentas para o uso da UML (ALMEIDA; DAROLT, 2001).

No decorrer do ano de 1996, determinadas companhias constituíram uma aliança de sócios da UML, e adotaram a UML como estratégia para seus negócios e ficaram dispostas a colaborar para a definição da UML. Espontaneamente estas companhias ofereceriam suporte para afeiçoar a UML como padrão para linguagens de modelagem no OMG (SILVA, 2007). Ainda hoje, devido a essa necessidade de criação da UML empresas e profissionais liberais da área estão ampliando estudos para melhor aplicá-la. Com isso, será apresentada na próxima seção, os campos em que essa ferramenta pode ser aplicada.

4.1.4 Aplicação

A UML é utilizada no desenvolvimento dos mais diferentes tipos de sistemas. Ela compreende sempre qualquer característica de um sistema em um de seus diagramas e é também aproveitada em diversas fases do desenvolvimento de um sistema, desde a especificação da análise de requisitos até a finalização com a fase de testes (FERREIRA et al., 2006).

Para Lacerda (2005) o objetivo da UML é delinear qualquer tipo de sistema, em termos de diagramas orientado a objetos. Naturalmente, a utilização mais comum é para criar modelos de sistemas de software, mas a UML também é empregada para representar sistemas mecânicos sem nenhum software. No Quadro 2 são mostrados alguns tipos diferentes de sistemas com suas características mais comuns:

Quadro 02 - Tipos de sistemas e suas características

TIPO DE SISTEMA	CARACTERÍSTICAS
Sistemas de Informação	Armazenar, pesquisar, editar e mostrar informações para os usuários. Manter grandes quantidades de dados com relacionamentos complexos, que são guardados em bancos de dados relacionais ou orientados a objetos.
Sistemas Técnicos	Manter e controlar equipamentos técnicos como de telecomunicações, equipamentos militares ou processos industriais. Eles devem possuir interfaces especiais do equipamento e menos programação de software de que os sistemas de informação. Sistemas Técnicos são geralmente sistemas real-time.
Sistemas Real-time Integrados	Executados em simples peças de hardware integrados a telefones celulares, carros, alarmes etc. Estes sistemas implementam programação de baixo nível e requerem suporte real-time.
Sistemas Distribuídos	Distribuídos em máquinas onde os dados são transferidos facilmente de uma máquina para outra. Eles requerem mecanismos de comunicação sincronizados para garantir a integridade dos dados e geralmente são construídos em mecanismos de objetos como CORBA, COM/DCOM ou Java Beans/RMI.
Sistemas de Software	Definem uma infraestrutura técnica que outros softwares utilizam. Sistemas Operacionais, bancos de dados, e ações de usuários que executam ações de baixo nível no hardware, ao mesmo tempo que disponibilizam interfaces genéricas de uso de outros softwares.
Sistemas de Negócios	descreve os objetivos, especificações (pessoas, computadores etc.), as regras (leis, estratégias de negócios etc.), e o atual trabalho desempenhado nos processos do negócio.

FONTE: Dantas (2001)

É fundamental compreender que a maior parte dos sistemas não possui apenas uma destas especialidades acima relacionadas, mas diversas delas ao mesmo tempo, e se tratando em software livre vê-se que o mesmo apresenta alguns dos atributos citados, trazendo-se a

ideia que esse tipo de abordagem de desenvolvimento também pode ser auxiliada com a UML (GUEDES, 2009). Para o autor a UML por ser uma linguagem padrão suporta modelagens de todos estes tipos de sistemas, que são representados através dos seus inúmeros diagramas, que serão mostrados de modo sucinto na próxima seção.

4.2 SUBCONJUNTOS DA UML

Na seção 4.1, a UML foi descrita como linguagem padrão para a modelagem de sistemas de software. A seguir, é apresentada uma breve descrição dos principais elementos e diagramas que formam o subconjunto utilizado na estrutura deste trabalho.

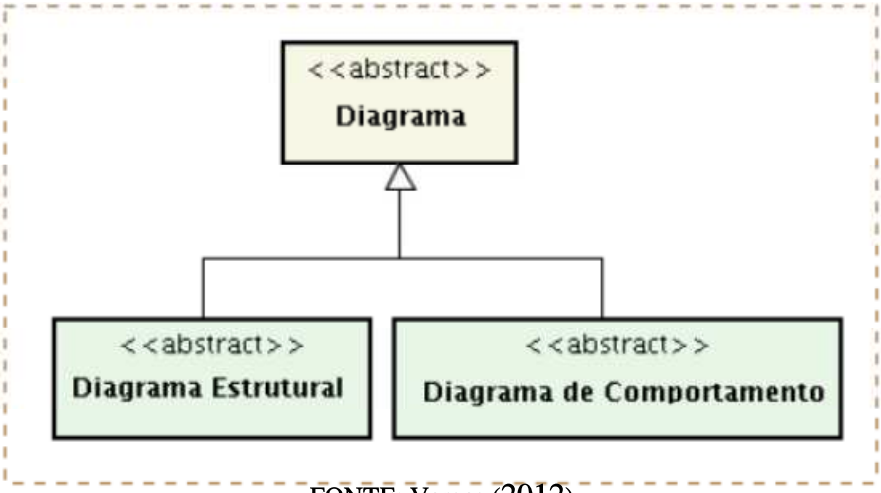
A UML mostrar-se constituída em duas amplas categorias: elementos estáticos e dinâmicos. Dos estáticos fazem parte os mecanismos de extensão (estereótipo, restrição, propriedades), tipos de dados, a base de construtores estáticos fundamentais e os modelos estruturais representados pelos diagramas de classes, objetos, componentes e deployment. Dos dinâmicos fazem parte às colaborações, casos de uso, máquinas de estados e grafos de atividades. A linguagem oferece inúmeros tipos de diagramas, e vários elementos sintáticos para ampla modelagem (DANTAS, 2001). No escopo deste trabalho, foi estudado de modo contextualizado apenas um subconjunto da UML, considerado essencial e simples o suficiente para sua utilização na modelagem de documentação.

Essencialmente, a UML admite que desenvolvedores visualizem os produtos de seu trabalho em diagramas unificados e padronizados, que junto com uma notação gráfica, a UML também especifica significados, isto é, semântica (ALMEIDA; DAROLT, 2001).

De um modo geral, um diagrama é uma representação gráfica de um sistema sob determinada visão. Existem diversos motivos para utilizá-los (CRUZ, 2006). O autor ainda continua dizendo que eles são benéficos na documentação, deixando a manutenção mais simplificada, auxiliam a entender melhor os requisitos de um sistema antes de implementá-lo, simplificam o entendimento de sistemas abstratos utilizando a técnica de dividir para conquistar.

Os diferentes diagramas da UML podem ser decompostos em dois grandes grupos, os diagramas estruturais e os comportamentais. Será mostrado um pouco de cada um desses grupos de diagramas no decorrer desta seção. A Figura 2 mostra a estrutura das categorias usando a notação de diagramas de classes (OMG, 2010).

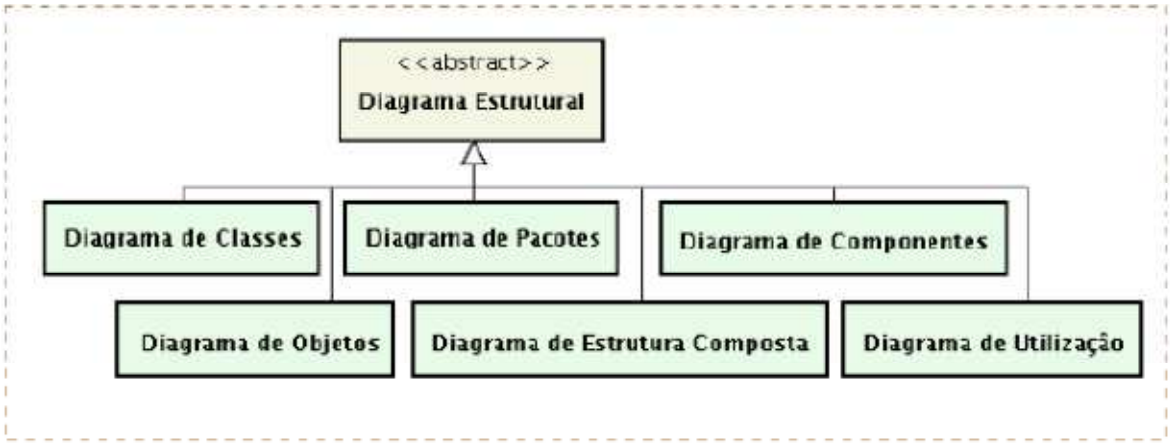
Figura 02 - Organização Geral dos Diagramas da UML



FONTE: Vargas (2012)

Os diagramas estruturais, exibidos na Figura 3 conforme a especificação OMG (OMG, 2010), abordam o aspecto estrutural tanto do ponto de vista do sistema quanto das classes. São usados para especificar, visualizar, construir e documentar os aspectos estáticos de um sistema, ou seja, a representação de seu esqueleto e estruturas relativamente estáveis (VARGAS, 2012). Os aspectos estáticos de um sistema de software compreendem a existência e a colocação de itens como classes, interfaces, colaborações, componentes.

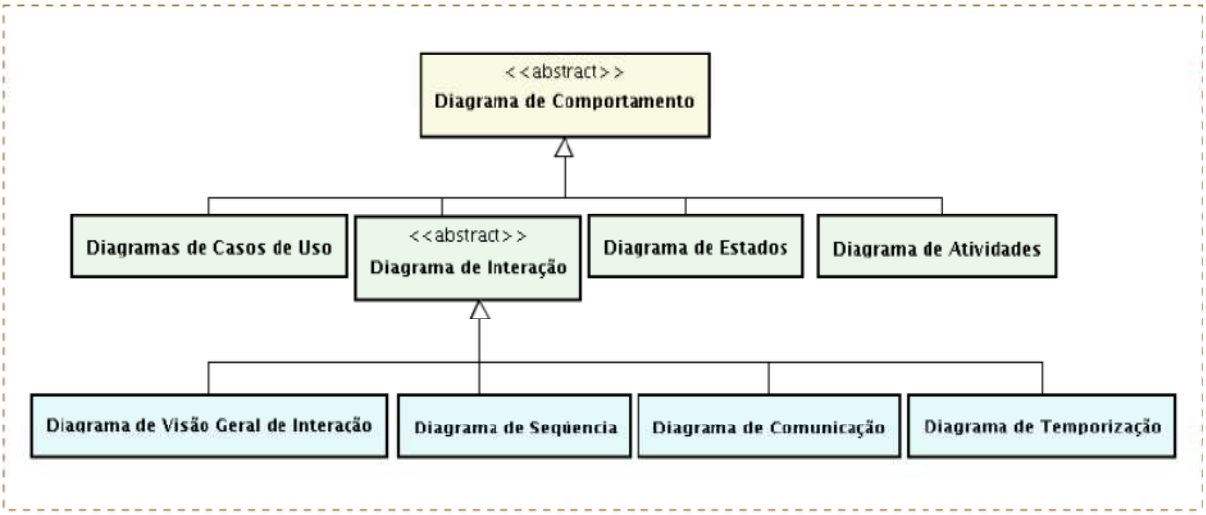
Figura 03 - Diagramas Estruturais



FONTE: Vargas (2012)

Os diagramas de comportamento, ilustrados na Figura 4 conforme a especificação OMG (OMG, 2010), são destinados a delinear o sistema computacional modelado quando em execução, isto é, como a modelagem dinâmica do sistema. São utilizados para especificar, visualizar, construir e documentar os aspectos dinâmicos de um sistema que é a representação das partes que passam por modificações, como por exemplo, o fluxo de mensagens ao longo do tempo e a movimentação física de componentes em uma rede.

Figura 04 - Diagramas Comportamentais



FONTE: Vargas (2012)

Cada tipo de diagrama prende uma perspectiva diferente e um determinado elemento poderá existir em múltiplos diagramas (BOOCH, 2000). Logo abaixo serão sintetizadamente apresentados as particularidades dos diagramas que compõem a linguagem UML.

Segundo Dantas (2001) os diagramas sugeridos pela UML são:

- **diagrama de classes**: mostra conjunto de classes, interfaces e colaborações, bem como seus relacionamentos. Esses diagramas são constantemente achados em sistemas de modelagem orientados a objeto e envolvem uma visão estática da estrutura do sistema. Os diagramas de classes abarcam classes ativas direcionam a perspectiva do processo estático do sistema;
- **diagrama de objetos:** expõe um conjunto de objetos e seus relacionamentos. Representa retratos estáticos de instâncias de componentes encontrados em diagramas de classes. São diagramas que compreendem a visão estática da estrutura ou processo de um sistema, como ocorre nos diagramas de classes, mas sob perspectiva de casos reais ou de protótipos;
- **diagrama de pacotes:** é um elemento sintático destinado a conter elementos sintáticos de uma especificação orientada a objetos. Seu intuito é discutir a modelagem estrutural do sistema dividindo o modelo em divisões lógicas e descrevendo as interações entre ele em um nível maior;
- **diagrama de estrutura composta:** provê meios de determinar a estrutura de um elemento e de focalizá-la no detalhe, na construção e em relacionamentos internos. O diagrama introduz a noção de "porto", um ponto de conexão do elemento modelado, a quem podem ser agregadas interfaces, usa a noção de "colaboração", que incide em

um conjunto de elementos interligados através de seus portos para a execução de uma funcionalidade específica – recurso de utilidade para a modelagem de padrões de projeto;

- **diagrama de componente:** apresenta as organizações e as dependências existentes em um conjunto de componentes. Envolve a visão estática da implementação de um sistema. Está relacionado aos diagramas de classes, pois caracteristicamente os componentes são mapeados para uma ou mais classes, interfaces ou colaborações;
- **diagrama de implantação:** exibe a configuração dos nós de processamento em tempo de execução e os componentes neles existentes. Compreende a visão estática do funcionamento de uma arquitetura diagramas de implantação. Está ligado aos diagramas de componentes, pois tipicamente um nó abrange um ou mais componentes;
- **diagrama de casos de uso:** expõe um conjunto de caso de uso e atores (um tipo exclusivo de classe) e seus relacionamentos. Diagramas de caso de uso abarcam a visão estática de casos de uso do sistema. Esses diagramas são fundamentais especialmente para a organização e a modelagem de comportamentos do sistema;
- **diagrama de interação:** mostra uma interação, consistindo de um conjunto de objetos e seus relacionamentos, contendo as mensagens que podem ser trocadas entre eles. Diagramas de interação compreendem a visão dinâmica do sistema. Composto por diagrama de sequências e diagrama de colaborações descritos a seguir;
- **diagrama de sequências:** mostra a troca de mensagens entre múltiplos objetos, em uma situação específica e delimitada no tempo. Coloca ênfase especial na ordem e nos momentos nos quais mensagens para os objetos são emitidas.
- **diagrama de máquina de estados:** ostentam uma máquina de estados, constituída por estados, transições, eventos e atividades. Os diagramas de estados abarcam a visão dinâmica de um sistema. São fundamentais principalmente para a modelagem de comportamentos de uma interface, classe ou colaboração e para dar destaque a comportamentos de um objeto ordenados por eventos;
- **diagrama de comunicação:** é destinado a delinear objetos interagindo e seus principais elementos sintáticos são "objeto" e "mensagem". Corresponde a um formato alternativo para narrar interação entre objetos. Vale lembrar que tanto o diagrama de comunicação como o diagrama de sequência são diagramas de interação;

- **diagrama de atividade:** é um modelo essencial de diagrama de gráficos de estado, expondo-se o fluxo de uma atividade para outra no sistema. Compreende a visão dinâmica do sistema e é fundamental principalmente para a modelagem da função de um sistema e dá ênfase ao fluxo de controle entre objetos;
- **diagrama de visão geral de integração:** é uma modificação do diagrama de atividades, recomendado na versão atualizada de UML. Seus elementos sintáticos são os mesmos do diagrama de atividades. As interações que fazem parte do diagrama de visão geral de interação podem ser referências a diagramas de interação existentes na especificação tratada;
- **diagrama de temporização:** consiste na modelagem de restrições temporais do sistema. É um diagrama originado no lançamento da segunda versão de UML, rotulado como diagrama de interação. Este diagrama modela interação e evolução de estados;
- **diagrama de colaborações:** é um diagrama de interação cujo destaque está na organização estrutural dos objetos que enviam e recebem mensagens.

As visões do sistema são descritas em uma quantidade de diagramas que retém informação destacando um aspecto particular do sistema (BOOCH, 2000). Analisando-se o sistema por meio de diversas visões, é possível se concentrar em um aspecto de cada vez. Para tanto, usar esses diagramas em conjunto com os demais recursos da UML na documentação SL, traz inúmeros benefícios que serão apresentados na próxima seção.

4.3 USANDO UML PARA DOCUMENTAR SOFTWARE LIVRE

Esta seção busca oferecer um panorama geral da importância do uso de modelos nas atividades de desenvolvimento de software, bem como a forma de utilização da UML na documentação no contexto de Software Livre.

4.3.1 Introdução

A UML é uma linguagem para especificação, visualização, documentação e construção que permite expressar modelos, principalmente orientados a objetos (LACERDA, 2005). Entretanto não indica como o trabalho deve ser realizado.

O emprego da UML em grandes projetos de Software Livre pode auxilia-lo a tornar-se mais eficiente, mas para tal precisa-se seguir um método de desenvolvimento (ALMEIDA; DAROLT, 2001). Para os autores o método de desenvolvimento seguido deve descrever uma quantidade de atividades a ser seguidas em uma ordem definida. Para obter com êxito os objetivos indicados pelo software, precisar-se ter bem determinado à expectativa do projeto.

Para Sanches (2001) um método tradicional de desenvolvimento orientado a objetos é dividido em análise de requisitos, análise, projeto, implementação e teste (BARROS, 2001), que é uma visão técnica do desenvolvimento. O RUP (Rational Unfied Process) é um outro método, desenvolvido pela Rational Inc. que é dividido em concepção, elaboração, construção e transição, que mostra uma visão mais gerencial do desenvolvimento (BOOCH, 2000). Existem inúmeros métodos de desenvolvimento de software que possuem abordagens diferenciadas.

No campo de Software Livre , o processo de desenvolvimento não é realizando seguindo as normas mais tradicionais existentes na engenharia de software, pois a produção de SL é considerada inovadora comparada aos métodos tradicionais, essa nova metodologia pode ser criada de acordo com as particularidades de cada sistema, e isso se dar pela flexibilidade encontrada nos Softwares Livres. Apesar de não se ter uma metodologia especifica para os SL, eles podem ser produzidos adotando métodos que priorizem a sua documentação, de preferencia aqueles que utilizam ferramentas para enriquecer essa etapa tão importante do processo de desenvolvimento.

A UML por ser considerada uma ferramenta ampla e moderna, que suporta a metodologia de desenvolvimento de sistemas livres, além das linguagens de modelagem e programação (FOWLER, 2004). A partir deste ponto a produção de Software Livre pode tomar como base para seu desenvolvimento uma metodologia que possua uma visão técnica da documentação que utiliza a UML. No decorrer desta seção, será apresentado de modo simples como a UML é utilizada para dar mais qualidade à documentação de SL.

4.3.2 Implementação da UML na documentação de Software Livre

O processo de documentação de um Software Livre começa desde as reuniões iniciais para a escolha do sistema a ser desenvolvido até a utilização do software finalizado (GAROFALO; CAMPOS, 2010). Para tanto, a utilização da UML nessa documentação inicia-se com o uso de determinados diagramas ainda na fase de análise de requisitos. Esta fase preocupa-se com as abstrações e mecanismos iniciais que se encontrarão presentes no

domínio do problema do software, os diagramas usados são os de classes em conjunto com diagramas de objetos, os diagramas de interação que se divide em sequência e colaboração, e também dos diagramas de estado e atividade, a utilização desses modelos é feita com o uso ferramentas adequadas (ALMEIDA; DAROLT, 2001).

Assim que se é feita esta modelagem inicial, simplifica-se a realidade para uma melhor compreensão do sistema em desenvolvimento. Para Fowler (2004) a UML cria seus modelos a partir de blocos distintos tais como: classes, interfaces, colaborações, componentes, dependências, generalizações, associações, etc. O uso dos diagramas são formas utilizadas para visualização de blocos de construção do próprio software (BOOCH, 2000).

Com essa modelagem inicial são três os documentos iniciais propostos por Medeiros (2004) como alicerces para a construção dos modelos: Visão, Nomenclatura e Glossário. O Quadro 3 mostra as definições destes documentos iniciais.

Quadro 03 - Tipos de documentos iniciais e suas definições

TIPO DO DOCUMENTO	DEFINIÇÃO
Documento Visão	O Documento Visão delineia, textualmente, aspectos relacionados à finalidade, tecnologia, linguagem, restrições e banco de dados da aplicação, além dos requisitos de documentação, e, graficamente, um modelo conceitual, abstraído por um diagrama de casos de uso. Vários autores indicam uma descrição sucinta e em uma linguagem de alto nível nesse documento, alertando para não transformá-lo em uma especificação demorada e detalhada do sistema.
Documento Nomenclatura	O Documento Nomenclatura é adquirido através de pequenas interferências no código-fonte, examinando e expondo o padrão usado para nomeação de palavras no corpo do mesmo. É importante lembrar que, de acordo com a história, sistemas em que não existe documentação, também não vão existir organização adequada, no que se refere à nomenclatura. A constatação dessa, torna inviável a confecção do documento nomenclatura, sem levar em consideração técnicas de refatoração.
Documento Glossário	O Documento Glossário explica os termos empregados no desenvolvimento e modelamento do software, sendo crucial para organizações que possuem alta rotatividade e que almejam minimizar o custo do aprendizado nas modificações.

FONTE: Medeiros (2004)

Baseado nestas primeiras análises e na produção dessas documentações iniciais trata-se então a fase de soluções técnicas. Para essas soluções são utilizados os diagramas de implementação, os quais estão decompostos em diagramas de componentes e implantação, esses diagramas são utilizados de acordo com suas características e funções (ALMEIDA; DAROLT, 2001).

Posteriormente, a implementação que é, na prática, a construção física do sistema proposto (SILVA, 1998). É onde todos os modelos criados no projeto são traduzidos para código que a máquina possa distinguir. É de fundamental importância que as fases anteriores já se encontrem finalizadas, pois a codificação é uma consequência natural do projeto.

Dentro dessa etapa de implementação o processo de tradução acontece quando o compilador aceita o código-fonte como entrada e gera um código-objeto como saída (PETERS; PEDRYCZ, 2001). Os autores ainda afirmam para que o código-fonte não seja gerado de maneira desigual do projeto, precisa-se ter todos os detalhes do projeto bem preparados e analisados.

A fim de que esse código-objeto seja gerado com sucesso, torna-se indispensável a seleção de uma linguagem de codificação, se a escolha for por uma linguagem abstrata ou pouco conhecida pode levar a códigos desorganizados e de difícil legibilidade (PRESSMAN, 2002). O autor diz que se precisa concentrar em escolher linguagens de codificação que tenha base ao projeto. A escolha de uma linguagem de programação para um projeto específico deve-se levar em consideração as suas facilidades e aonde ela irá auxiliar o melhor desenvolvimento do projeto.

Após se ter originado o código-objeto por intermédio da linguagem de codificação é gerada a documentação do código-fonte. Um código-fonte bem estruturado e documentado será de fundamental importância para futuras alterações no sistema (BUBACK, 2007). Para o autor, a estruturação e documentação iniciam-se com a seleção de nomes identificadores (variáveis e rótulos), continuando com a colocação e composição dos comentários e por fim com a organização visual do código.

Com tudo isso, o uso da UML na documentação de SL pode acontecer em paralelo com as demais etapas do processo de desenvolvimento de software. Para Fowler (2004) o uso dessa ferramenta resulta inúmeros benefícios, a exemplo, os diagramas que são usados para produzir a documentação, apresentam uma linguagem de fácil manipulação; a modelagem inicial do sistema a ser desenvolvido é feita de acordo com as particularidades de cada grupo de desenvolvimento; outro beneficio é abstração da complexidade do sistema em desenvolvimento através dos inúmeros recursos oferecidos pela UML.

Ainda que a UML tenha oferecido vários e inquestionáveis benefícios buscando especificar, padronizar, simplificar, estruturar e documentar os processos de desenvolvimento de software, independente da linguagem de programação ou plataforma de desenvolvimento, a UML ainda possui algumas lacunas como o problema relacionado à semântica, onde diferentes desenvolvedores podem diagramar um mesmo sistema de modos distintos

(SOUSA, 2012). Na concepção do autor, na essência do princípio do UML, um mesmo sistema precisaria ser diagramado de apenas um único modo, seja pelo desenvolvedor A, seja pelo desenvolvedor B. Todavia esse objetivo ainda não é possível no contexto atual da UML. Quem sabe, refletindo nisso os criadores do UML deixaram espaço para que no futuro a questão da semântica do UML seja decidida.

Diante de tudo que foi exposto, pode-se perceber que a utilização da UML na documentação vem a somar com as demais documentações e artefatos que são produzidas no processo de desenvolvimento de SL. Neste contexto, a proposta de adotar a UML na documentação poderá seguir suas etapas de acordo com o fluxo de desenvolvimento de um projeto livre.

4.4 CONCLUSÕES DO CAPÍTULO

Com o desenvolvimento de sistemas cada vez mais complexos, a utilização de diferentes formas e ferramentas torna-se fundamental para auxiliar esse processo. Nesse sentindo, foi apresentado no transcorrer deste trabalho, o uso da ferramenta UML como forma de ajudar a produção dos artefatos que são desenvolvidos na documentação de um software.

A documentação gerada a partir de modelos de documentos é automaticamente atualizada quando qualquer um dos objetos é modificado, tendo em vista que a ferramenta se mostra flexível para ser aplicada em diferentes contextos de desenvolvimento de SL, o que acaba por reduzir o esforço necessário de atualização e consequentemente das modificações, e também permiti uma maior integração entre as ferramentas e artefatos que se encontram no desenvolvimento de software.

Por fim, a UML se apresenta como uma linguagem padronizada para a construção de modelos que proporciona um maior entendimento do sistema que está sendo desenvolvido, então utilizar esse recurso de forma consciente resulta em criar documentos com mais qualidade, por isso à importância destes modelos para o sucesso de desenvolvimento.

CAPÍTULO 5 – CONSIDERAÇÕES FINAIS

Neste capítulo procura-se destacar as contribuições do trabalho para a comunidade de software em geral e, mais especificamente, para os envolvidos no desenvolvimento de Software Livre. Apesar de ter alcançado o principal objetivo previamente imaginado, algumas questões ficaram em aberto e novas perspectivas surgiram para trabalhos futuros relacionados.

O desenvolvimento de Software Livre (SL) tem evoluído bastante, proporcionando o surgimento de uma quantidade cada vez maior de projetos de SL e contraindo importância considerável no cenário mundial de desenvolvimento de software. O SL ocasionou à comunidade de software, uma nova forma de se desenvolver software, diferente da tradicional, originando mudanças culturais, políticas e econômicas.

A partir de uma análise sobre o cenário atual do desenvolvimento de software livre, constatou-se que dificilmente há projetos de SL desenvolvido com a utilização de processos baseados em normas e modelos de qualidade mundialmente reconhecidos. Com isso, os documentos que são produzidos no decorrer do desenvolvimento desses softwares muitas vezes se apresentam incompletos e mal elaborados ou até mesmo deixam de existi pela informalidade utilizada nessa modalidade de software.

O software livre é considerado uma produção colaborativa de divulgação ampla, então se conclui que documentar esse processo é fundamental, assim como divulgar os resultados, para que o princípio de compartilhar conhecimento seja cumprido. Por tanto, os software que apresentam algum tipo de documentação no seu processo de desenvolvimento, mostrar-se diferenciais que resultam na qualidade do próprio software.

Após considerável estudo bibliográfico sobre o processo de desenvolvimento de software livre, mais especificamente na etapa de produção de documentação, pôde-se perceber que esta modalidade de sistema necessita de uma documentação mais técnica que possa ser utilizada para facilitar a compreensão do sistema e as suas possíveis modificações.

Assim o objetivo do trabalho foi mostrar que a ferramenta de modelagem UML pode fazer parte da documentação de software livre e auxiliar na construção de uma documentação com mais qualidade, pois com ela pode ser elaborada a estrutura de sistemas complexos de software, simplificando a realidade dos sistemas, além de tornar possível a construção de modelos para compreender melhor o sistema que está sendo desenvolvido.

Em virtude das constantes inovações que surgem em ferramentas de desenvolvimento de software livre, pode-se afirmar a importância do uso dessa metodologia que engloba todas

as fases do processo, desde os eventos iniciais, passando pela análise de requisitos, análise, projeto, implementação e testes.

Com o uso dessa metodologia pode ser destacada a importância de manter a documentação atualizada continuamente, tornando o processo de documentar sistemático e dinâmico, conforme a solicitação de alterações. Por fim, tem-se em mente que a formalização da documentação deve passar a integrar, de forma obrigatória, o processo de desenvolvimento e manutenção de software.

Para tanto, com o desenvolvimento de sistemas mais complexos, o uso de uma metodologia unificada, proporciona uma comunicação maior entre toda a equipe envolvida (cliente, analista, programador, etc), e uma documentação relevante.

Neste contexto, o trabalho mostrou a importância da utilização de uma metodologia de desenvolvimento da documentação de software baseada em UML, a partir da qual o desenvolvedor pode seguir seus passos para o andamento adequado do desenvolvimento de um projeto.

Após o estudo da ferramenta UML como componente da documentação de software livre, esta ferramenta mostrou-se bastante complexa e eficaz, pois com ela se pode produzir uma documentação interativa e automática dos processos de modelagem, permitindo até mesmo gerar código-fonte a partir dos seus modelos.

Com o desenvolvimento deste trabalho utilizando esta metodologia de utilizar a UML, afirma-se que os resultados esperados e as hipóteses levantadas foram alcançados, tendo em vista que a ferramenta pode facilmente ser aplicada nos mais diferentes contextos de desenvolvimento de software livre e que esta atende todas as fases de desenvolvimento.

Desta forma, adotar a UML de forma consciente e madura significa dar um passo importante em direção à criação de aplicações mais completas e com mais qualidade.

5.1 LIMITAÇÕES DO TRABALHO

Durante o desenvolvimento deste trabalho, algumas limitações foram identificadas, dentre elas, pode-se citar a necessidade de se aplicar a metodologia de utilizar a UML em algum processo proposto em projeto piloto no intuito de avaliá-los e melhorá-los. Para que se pudesse efetivar essa aplicação, seria necessário um tempo maior que, comumente, não é disponível à elaboração de um trabalho de conclusão de curso de graduação.

Mais uma limitação encontrada foi os materiais que são usados como referenciais teóricos, principalmente os que abordam o assunto principal da pesquisa - modelagem UML,

o que tornou muito difícil de encontrar materiais de qualidade que pudessem dar mais embasamento a este pesquisa.

Outra limitação enfrentada neste trabalho foi à impossibilidade de se construir um protótipo de documentação para mostrar na prática o uso da UML e, assim, averiguar se todos os requisitos levantados no decorrer do trabalho são suficientes para atender de forma eficaz aos objetivos do projeto.

5.2 RECOMENDAÇÕES PARA TRABALHOS FUTUROS

Por ser uma área em expansão e de grande importância para a Engenharia de Software, muito pode ser feito no contexto da documentação de software para dar continuidade ao trabalho realizado.

Embora a revisão bibliográfica venha mostrar a importância do uso da ferramenta UML na documentação de software livre, resultando na principal contribuição deste trabalho, identificam-se outros aspectos que podem ser evoluídos, mas que não foram discutidos no contexto deste trabalho. Dentre as possíveis melhorias, pode-se citar:

- Aplicação da UML em projetos piloto para que sejam feitos os ajustes necessários, aperfeiçoando-os a ponto de serem realmente disponibilizados para apoiarem o desenvolvimento de Software Livre com uma documentação de qualidade.
- Estudo de novas ferramentas livres que utilizam a UML.
- Desenvolver novas pesquisas visando corrigir os pontos-fracos encontrados.
- Desenvolvimento de uma ferramenta UML baseando-se na metodologia desenvolvida, para contextos específicos de produção de software livre.
- Utilização da metodologia proposta no apoio de diferentes softwares livres.

Neste sentido, é importante evoluir as pesquisas relacionadas à como utilizar de modos diferentes a modelagem UML no desenvolvimento da documentação de Software Livre. Há uma expectativa de que os benefícios decorrentes do uso dessa ferramenta de apoio ao desenvolvimento da documentação seriam muito maiores se a própria ferramenta pudesse ser usada integrada às demais ferramentas livres.

REFERÊNCIAS

ALMEIDA, A.; DAROLT, R. **Pesquisa e Desenvolvimento em UML**. Universidade do Sul de Santa Catarina. Araranguá. 2001. Disponível em: < http://www.oodesign.com.br/blog/>. Acesso em: 10 de março 2012.

AMBLER, S. W. **Agile Documentation**. The Official Agile Modeling (AM) Site. 2001. Disponível em: <http://www.agilemodeling.com/essays/ agileDocumentation.htm>. Acesso em: 02 abr. 2012.

BARROS, P. **UML : Linguagem de Modelagem Unificada em Português**. 2001. Disponível em:<http://cc.usu.edu/~slqz9/uml>. Acesso em: 13 mar. 2012.

BERTOLLO, G. **Definição de Processos em um Ambiente de Desenvolvimento de Software.** 2006. 108f. Dissertação (Mestrado). Universidade Federal do Espírito Santo, Vitória-ES.

BOOCH G. et al. **UML : Guia do Usuário, O mais avançado tutorial sobre Unified Modeling Language**. Rio de Janeiro: Editora Campus. 2000.

BRITO, R.; FERREIRA, P.; SILVA, K.; BURERGIO, V.; LEITE, I. **Uma experiência na implantação de processo em uma fábrica de software livre**. VI Simpósio Internacional de Melhoria de Processos de Software, São Paulo, SP – Brasil. 2004. Disponível em: < http://www.cin.ufpe.br/~in953/publications/papers/UmaExperienciaNaImplantacaoDeProcessoEmUmaFabricaDeSoftwareLivre.pdf >. Acesso em: 12 fev. 2012.

BUBACK, S. N. **ODEDoc: Uma Ferramenta de Apoio à Documentação no Ambiente ODE**. 2007. Monografia (Graduação). 68f. Universidade Federal do Espírito Santo. Vitória – ES.

CARVALHO, J. C. F. F.; AMARAL, A. M. M. M. **Estudo e Definição da Aplicação para Controle de Versões dos Artefatos Gerenciados pela Ferramenta S.A.Do.M (Software Artifacts Documentation and Management**). 2010. V Mostra Interna de Trabalhos de Iniciação Científica. CESUMAR – Centro Universitário de Maringá. Maringá – Paraná. Disponível em:< http://www.cesumar.br/prppge/pesquisa/mostras/quin_mostra/julio_cezar_fialho_freire_carvalho.pdf>. Acesso em: 04 abr. 2012.

CHRISTOPH, R. H. **Engenharia de software para software livre**. 2004. 118f. Dissertação (Mestrado). Departamento de Informática, Pontifica Universidade Católica do Rio de Janeiro, Rio de Janeiro.

CIOCH, F. A.; PALAZZOLO, M.; LOHRER S. **A Documentation Suite for Maintenance Programmers**. Proceedings of the International Conference on Software Maintenance (ICSM '96). 1996.

COELHO, H. S. **Documentação de Software: Uma necessidade**. Revista Texto Livre: Linguagem e Tecnologia. 2009. Disponível em: <

www.periodicos.letras.ufmg.br/index.php/textolivre/article/.../24/24 >. Acesso em: 19 fev. 2012.

COSTA, R. C.; SANTOS, R. F. O. **Conhecendo o Software Livre**. Universidade, EAD e Software Livre, Evento online Assíncrono. Revista Texto Livre. 2010. Disponível em: <http://ww.textolivre.pro.br/blog/UEADSL/2010_2/artigosPDF/index.php>. Acesso em: 12 fev. 2012.

CRUZ, E. R. **Inspeção de Especificações de Requisitos Representadas em Unified Modeling Language (UML)**. 2006. 113f. Universidade Metodista de Piracicaba – UNIME, Piracicaba – SP.

DANTAS, A. R. **ORÁCULO: Um Sistema de Críticas para a UML**. 2001. 53f. Monografia (Graduação). Departamento da Ciência da Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro-RJ.

DAVIS, M.; O'DONOVAN, W.; FRITZ, J. **Linux and open source in the academic enterprise**. Proc. of the 28th SIGUCCS Conference on User Services, Richmond, ACM New York. 2000. Disponível em: < http://dl.acm.org/citation.cfm?id=354923>. Acesso em: 23 fev. 2012.

DIPOLD, R. D. **Potencialidade econômica do software livre**. 2005. 65f. Monografia (Graduação). Universidade Estadual do Oeste do Paraná, Cascavel – PR.

DORNELAS, G. C. **A Viabilidade Econômica da Adoção do Software Livre no Contexto Universitário**. 2004. 53f. Monografia (Graduação). Universidade Federal de Viçosa – Minas Gerais.

ERIKSSON, H. **The semantic-document approach to combining documents and ontologies**. International Journal of Human-Computer Studies Volume 65, Issue 7. 2007. Disponível em:< http://www.sciencedirect.com/science/article/pii/S1071581907000468>. Acesso em: 04 abr. 2012.

FALBO, R. A. **Engenharia de Software: Notas de Aula**. 2005. 99f. Material de Aula. Universidade Federal do Espírito Santo, Vitória - ES. Disponível em: < http://www.inf.ufes.br/~falbo/download/aulas/es-g/2005-1/NotasDeAula.pdf >. Acesso em: 23 fev. 2012.

FERREIRA, P. P. D.; MOURA Jr., A. S.; CARVALHO, J. F. N. **Metodologia de construção da documentação de um sistema já existente através via UML**. X Seminário de Automação de Processos - ABM, Belo Horizonte - MG. 2006.

SALLES FILHO, S. S.; STEFANUTO, G. N.; ALVES, A. M.; DELUCCA, J. **O impacto Software Livre e de Código Aberto (SL/CA) nas condições de apropriabilidade na indústria de software brasileira**. XI Seminario de Gestión Tecnológica - ALTEC 2005, 2005, Salvador - BA. 2005.

FORWARD, A. **Software Documentation – Building and Maintaining Artefacts of Comunication**. 2002. Dissertação (Mestrado). Universidade de Ottawa, Ottawa, Toronto, Canadá.

FORWARD, A.; LETHBRIDGE, T. C. **The relevance of software documentation, tools and technologies: a survey**. Proceedings of the 2002 ACM Symposium On Document Engineering. 2002. Disponível em:< http://www.literateprogramming.com/doceng.pdf>. Acesso em: 04 abr. 2012.

FOWLER, M. **UML Essencial**. 3ª. ed. São Paulo: Bookman. 2004.

FREEMAN, R. M.; MUNRO, M. **Redocumentation for the Maintenance of Software**. Proceedings of the 30th annual Southeast regional conference, ACM. New York. 1992. Disponível em: < http://dl.acm.org/citation.cfm?id=503765>. Acesso em: 10 fev. 2012.

FREE SOFTWARE FOUNDATION, **O que é software livre?**. 2012. Disponível em: <http://www.gnu.org/philosophy/free-sw.pt.html>. Acesso em: 23 mar. 2012.

FREITAS, D. N. **Metodologia de Desenvolvimento de Software Livre com Arquiteturas Orientadas a Serviços: Um Estudo de Caso em um Ambiente de Tradução Automática**. 2009. 79f. Dissertação (Mestrado). Universidade Federal do Paraná, Curitiba – PR.

FUGGETTA, A. **Software Process: A Roadmap**. Proc. of The Future of Software Engineering, ICSE'2000, Limerick, Ireland. 2000.

GAROFALO, S.; CAMPOS, G. A. M. **Cerceamento ou Acessibilidade: Uma Discussão sobre a Documentação em Software Livre**. Texto Livre, v. 3, p. 1-8. 2010.

GODFREY, M.W. **Evolution in Open Source Software: A Case Study**. Software Architecture Group (SWAG), Department of Computer Science, University of Waterloo ICSM. 2000. Disponível em: < http://flosshub.org/system/files/godfrey00.pdf >. Acesso em: 23 mar. 2012.

GUEDES, G. T. A. **UML 2: Uma Abordagem Prática**. São Paulo: Novatec Editora. 3ª Edição. 2009.

HCI JOURNAL. **What to put in software maintenance documentation**. HCi Consulting, 2001. Disponível em: <http://www.hci.com.au>. Acesso em: 04 abr. 2012.

HEXSEL, R. A. **Software Livre – Propostas de Ações do Governo para Incentivar o Uso de Software Livre**. Relatório Técnico – RT-DINF 004/2002, Universidade Federal do Paraná, Curitiba – PR. 2002.

JONHSON, K. **A Descriptive Process Model for Open-Source Software Developmen**t. Submitted to the Faculty of Graduate Studies in Partial Fulfillment of The Requirements for the Degree of Master Science, Galgary, Alberta. 2001.

KON, F.; SABINO, V. **Licenças de Software Livre: História e Classificação**. Congresso Internacional Software Livre e Governo EletrônicoArtigos CONSEGI 2009. 1ª ed. Brasília: Fundação Alexandre Gusmão. 2009. Disponível em:< http://ccsl.ime.usp.br/files/relatorio-licencas.pdf >. Acesso em: 12 fev.2012.

LACERDA, G. S. **FrameworkDoc: ferramenta de documentação e geração de artefatos de software**. 2005. 77f. Dissertação (Mestrado). Universidade Federal do Rio Grande do Sul, Porto Alegre - RS.

LEHMAN, M. M. **Laws of Software Evolution Revisited**. European Workshop on Software process Technology. Springer-Verlag London 1996. Disponível em: <http://dl.acm.org/citation.cfm?id=681473>. Acesso em: 10 fev. 2012.

LERNER, J.; TIROLE, J. **Some Simple Economics of Open Source**. The Journal of Industrial Economics, V. L, n. 2, p. 197-234. 2002.

LETHBRIDGE, T. C.; SINGER J.; FORWARD, A. **How Software Engineers Use Documentation: The State of the Pratice**. IEEE Software. 2003. Disponível em: <http://www.cs.duke.edu/courses/cps196.1/current/classwork/Lethbridge-Singer-Forward-2003.pdf >. Acesso em: 04 abr. 2012.

LIMA, C. A. A. **Práticas para gerência e desenvolvimento de projetos de software livre observadas em comunidade de sucesso**. 2005. 162f. Dissertação (Mestrado). Universidade Federal de Campina Grande, Campina Grande – PB.

MATTE, A. C. F. **Uma definição informal de documentação: análise semiótica**. Revista Texto Livre: Linguagem e Tecnologia, v. 2, nº 1, p. 1-16. 2008. Disponível em: <http://www.textolivre.net/revista/index.php/TextoLivre/article/viewFile/12/13>. Acesso em 04 abr. 2012.

MEDEIROS, E. S. **Desenvolvendo software com UML 2.0: definitivo**. São Paulo: Pearson Makron Books. 2004.

MICHELAZZO, P. **A Documentação de software**. 2006. Disponível em:<http://www.michelazzo.com.br/ node/123>. Acesso em: 04 abr. 2012.

MOURÃO, W. I. **MDA: Fazendo a UML valer a pena**. Blog do Valter. 2007. Disponível em: <http://waltermourao.com.br/export/waltermourao/download/MDA_-_Fazendo_a_UML_valer_a_pena.pdf> . Acesso em: 23 fev. 2012.

NAKAGAWA, E. Y. **Software Livre: Processo e Produto Livres no Desenvolvimento de Aplicações Web**. 2002. Tese (Doutorado). Instituto de Ciências Matemáticas e de Computação – ICMC/USP, São Carlos – SP.

NUNES, V. B. **Integrando Gerência de Configuração de Software, Documentação e Gerência de Conhecimento em um Ambiente de Desenvolvimento de Software.** 2005. 215f. Dissertação (Mestrado). Universidade Federal do Espírito Santo, Vitória - ES.

NUNES, V. B.; SOARES, A. O.; FALBO, R. A. **Apoio à Documentação em um Ambiente de Desenvolvimento de Software**. 7º Workshop Ibero-americano de Engenharia de Requisitos e Ambientes de Software, Arequipa - Perú. 2004.

OMG. **Unified Modeling Language: infrastructure**. 2010. Disponível em:< http://www.citeulike.org/user/knowlengr/article/8980295 >. Acesso em: 13 mar. 2012.

OUCHI, M. L. **Software Maintenance Documentation**. Proceedings of the 4th anual international conference on Systems documentation (SIGDOC'85), New York, EUA, ACM Press.1985.

PENDER, T. **UML - A Bíblia**. São Paulo: Editora Campus, 2004.

PETERS, J.; PEDRYCZ, W. **Engenharia de Software: Teoria e Prática**. Rio de Janeiro: Editora Campus. 2001.

PRESSMAN, R. S. **Engenharia de Software**. Rio de Janeiro: McGraw Hill, 5ª edição, 2002.

RAYMOUND, E. **The cathedral and the bazaar**. O'Reilly & Associates, 1999. Disponível em: <http://catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/>. Acesso em 20 mar. 2012.

REIS , C. R. **Caracterização de um Processo de Software para Projetos de Software Livre**. 2003. 247f. Dissertação (Mestrado). Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, São Carlos – SP.

ROCHA, A. R. C. **Qualidade de software. Teoria e prática**. São Paulo: Prentice Hall, 2001.

SAMPAIO, M. C. **História de UML**. 2007. Disponível em:<http://www.dsc.ufcg.edu.br/sampaio> Acesso em: 20 mar. 2012.

SANCHES, R. **Documentação de Software**. Qualidade de Software: Teoria e Prática, Eds. A.R.C. Rocha, J.C. Maldonado, K. Weber, Prentice Hall, 2001.

SANTOS, C. E. Jr. **Texto Livre em documentação no desenvolvimento de softwares em comunidades**. Evento Online Assíncrono. Universidade Federal de Minas Gerais. 2011. Disponível em: <http://ueadsl.textolivre.pro.br/2011.2/papers/upload/71.pdf>. Acesso em: 23 fev. 2012.

SOFTEX .**O impacto do software livre e de código aberto na indústria de software do Brasil.** Softex Campinas. 2005. Disponível em: <www.softex.br>. Acesso em: 23 mar. 2012.

SILVA, A. M. R.; VIDEIRA, C. A. E. **UML, Metodologias e Ferramentas CASE**. Coleção Tecnologias. Centro Atlantico Lda. Porto-Lisboa-Portugal. 1ª Edição. 2001.

SILVA, B. C. C. **Processos e Ferramentas para o Desenvolvimento De Software Livre: Um Estudo de Caso**. 2006. 93f. Dissertação (Mestrado). Universidade Federal do Espírito Santo, Vitória-ES.

SILVA, N. P. **Projeto e Desenvolvimento de Sistemas**. São Paulo: Érica. 1998.

SILVA, P. B. **Adequação da Ferramenta de Documentação de ODE a uma Ontologia de Artefatos**. 2004. Monografia (Graduação). Universidade Federal do Espirito Santos, Vitória - ES.

SILVA, R. P. **UML 2 em Modelagem Orientada a Objetos**. Florianopolis: Visual Books. 2007.

SILVEIRA, S. A. **Software livre: a luta pela liberdade do conhecimento**. São Paulo: Editora Fundação Perseu Abramo (Coleção Brasil Urgente). 2004.

SOURCEFORGE, Home Page. 2012. Disponível em: <http://sourceforge.net/index.php>. Acesso em: 12 jan. 2012.

SOUSA, M. M. **Modelagem de Software**. 2012. Disponível em: <http://marcosmoraisdesousa.blogspot.com.br/2012/04/modelagem-de-software.html> Acesso em: 20 mar. 2012.

SOUZA, S. C. B.; NEVES, W. C. G.; ANQUETIL, N.; OLIVEIRA, K. M. **Documentação essencial para manutenção de software II**. 1° Workshop de Manutenção de Software Moderna, 2004, Brasília-DF, Brasil. 2004.

SOUZA, S. C. B.; NEVES, W. C. G.; ANQUETIL, N.; OLIVEIRA, K. M. **Investigação da documentação de maior importância para manutenção de software**. Jornada Iberoamericanas en Ingeniería del Software e Ingeniería del Conocimiento, Madrid, Espanha. 2004.

VARGAS, T. C. S. **A história da UML e seus diagramas**. 2012. Disponível em:<http://projetos.inf.ufsc.br/arquivos_projetos/projeto_721/artigo.tcc.pdf> Acesso em: 20 mar. 2012.