CENTRO DE ENSINO SUPERIOR DE JUIZ DE FORA
THIAGO BELLOTTI FURTADO

EVOLUÇÃO DE SOFTWARE: FUNDAMENTOS, PROCESSOS E APLICAÇÃO

Juiz de Fora
2007

THIAGO BELLOTTI FURTADO

EVOLUÇÃO DE SOFTWARE: FUNDAMENTOS, PROCESSOS E APLICAÇÃO

Trabalho de Conlcusão de Curso (Monografia), apresentado ao Centro de Ensino Superior de Juiz de Fora, como requisito parcial para conlusão do Curso de Graduação em Sistemas de Informação.

Orientadora: Profª Alessandreia Marta de O. Julio

Juiz de Fora
2007

Ficha Catalográfica elaborada pela Biblioteca Esdeva – CES/JF
Bibliotecária: Alessandra C. C. Rother de Souza – CRB6-1944

FURTADO, Thiago Bellotti
Evolução de Software: fundamentos, processos e aplicação. [manuscrito] / Thiago Bellotti Furtado. – Juiz de Fora: Centro de Ensino Superior de Juiz de Fora, 2007.
46 f.

Monografia (graduação) – Centro de Ensino Superior de Juiz de Fora (MG), Curso de Sistemas de Informação.
"Orientadora: Alessandreia Marta de O. Julio"

1. Informática 2. Evolução de Software. I. Centro de Ensino Superior de Juiz de Fora. II. Título.

CDD – 005.1

FOLHA DE APROVAÇÃO

BELLOTTI, Thiago Furtado. Evolução de software: conceitos, processos e aplicação. Trabalho de Conclusão de Curso (Monografia), apresentado como requisito parcial à conclusão do curso Graduação em Sistemas de Informação, do Centro de Ensino Superior de Juiz de Fora, realizada no 2° semestre de 2007.

BANCA EXAMINADORA

_____

Prof. Ms. Alessandreia Marta de Oliveira Julio
Orientadora

_____

Prof. Geraldo Magela Dutra Gonçalves
Membro Convidado 1

_____

Prof. Ms. José Honório Glanzmann
Membro Convidado 2

Examinado(a) em: ____/____/_____.

Dedico este trabalho a Deus, por ter me dado saúde e a oportunidade para que o desenvolvesse. A meus pais, Gonzaga e Luciana, pelo amor e confiança, às minhas irmãs, Tatiana e Thaís pelo carinho e à minha namorada Ana Flávia pelo amor e apoio.

AGRADECIMENTOS

Agradeço a Deus, por ter me dado saúde e a oportunidade de realizar este trabalho.
Agradeço a minha família, principalmente a meus pais Gonzaga e Luciana que com muito esforço e trabalho me ajudaram com amor e confiança superar mais uma etapa de minha vida.
Agradeço aos amigos pelos momentos de descontração e incentivo. Também agradeço aos colegas de trabalho pelos ensinamentos e experiências compartilhadas. Agradeço a minha namorada Ana Flávia pelo carinho, amor e o grande apoio dado para realizar este trabalho.
Agradeço especialmente a Professora e orientadora Alessandreia Marta de Oliveira Julio que demonstrou paciência e profissionalismo na orientação, ajudando-me a traçar os caminhos a serem superados para realizar um trabalho de qualidade.
Agradeço ao Professor Marco Antônio Pereira Araújo por ter fornecido material para a elaboração deste trabalho.
Agradeço a todos que de uma maneira ou outra contribuíram para este trabalho.

“Bom mesmo é ir a luta com
determinação, abraçar a vida e viver com
paixão, perder com classe e vencer com
ousadia, porque o mundo pertence
a quem se atreve e A VIDA É MUITO
para ser insignificante”
Charles Chaplin

RESUMO

BELLOTTI, Thiago Furtado. **Evolução de software**: conceitos, processos e aplicação. 46 pf. Trabalho de Conclusão de Curso (Monografia – Graduação em Sistemas de Informação). Centro de Ensino Superior de Juiz de Fora, Juiz de Fora, 2007.

A manutenção de software está presente nos ciclos de evolução de um software. A partir do momento que um software começa a ser desenvolvido, manutenções já são feitas, porém mesmo o software estando totalmente pronto, não há garantia de que ele esteja funcionando perfeitamente. A evolução de software busca uma alternativa para que os softwares sejam cada vez mais independentes de tecnologias e mais perfeitos, procurando minimizar e simplificar a manutenção do software, sempre que necessário. Modificar um software ao longo do tempo, identificar as fases da evolução do software e os problemas que ocorrem durante as etapas da evolução é muito importante para adiar o seu fim. Porém, evoluir um software junto a manutenções não é um processo tão simples, mas perfeitamente possível. Este estudo mostra a importância de se adotar as técnicas de evolução de software e os pontos importantes que tornam o processo de evolução mais eficaz. Entender as causas do envelhecimento do software, e que atitudes devem ser tomadas para retardar esse processo.

Palavras-Chave: Evolução de Software; Manutenção; Rejuvenescimento.

ABSTRACT

The software maintenance is present in cycles of software evolution. Since the moment that a software begins to be developed, maintenance already are made, but even the software is fully ready, there isn't guarantee that it's perfectly working. The software evolution search alternative so the software's are increasingly independent of technologies and more perfects, seeking simplify and reduce the software maintenance, that always necessary. Modify the software over time, identify the stages the software evolution and the problems occurring during the evolution steps is very important to postpone your end. But, developing a software with maintenance it isn't a process as simple, but perfectly possible. This study show importance to adopt the techniques of software evolution and the important points that makes the process of development more efficient. Understand the causes of aging software, and that attitude should be taken to delay this process.

Keywords: Software Evolution; Maintenance; Rejuvenation.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

A manutenção de software está presente nos ciclos de evolução de um software. A partir do momento que um software começa a ser desenvolvido, manutenções já são feitas, porém mesmo o software estando totalmente pronto, não há garantia de que ele esteja funcionando perfeitamente.

Ao utilizar um sistema de informações, este deve ser adaptado continuamente para que não se torne progressivamente menos satisfatório. Por isso, para retardar o desgaste dos softwares é preciso planejar bem sua estrutura de forma que suas métricas e seus requisitos estejam de acordo com as funções que o mesmo desempenha (BARBOSA e PERKUSICH, 2006).

A necessidade de realizar modificações no software se torna cada vez mais constantes à medida que surgem novos objetivos e estes se modificam. Devido a grandes mudanças em sistemas de software, é necessário compreender como os sistemas sofrem mudanças, assim é mais fácil acompanhar sua manutenção evolutiva. Tendo o conhecimento de evolução, os riscos relacionados à perda de qualidade do software são menores. Mesmo assim não se pode garantir que um processo de evolução atinja o objetivo final sem perdas de qualidade. Ainda é preciso observar o comportamento e as tendências de um software, para prever riscos futuros na sua implementação (ARAÚJO e TRAVASSOS, 2006).

Por outro lado, a evolução busca uma alternativa para que os softwares sejam cada vez mais independentes de tecnologias e mais perfeitos, procurando minimizar e simplificar sua manutenção, sempre que necessário. É preciso prever o futuro de um software para que assim, o mesmo esteja preparado para o presente.

A escolha do tema deve-se primeiramente ao fato de que a evolução de software está distante do conhecimento das pessoas ou ainda não é empregada de modo eficiente, contribuindo para o fim da vida de um software. A evolução é muito importante para garantir que um software tenha uma vida mais longa obtendo dessa forma uma estrutura mais robusta capaz de superar ambientes que sofram modificações ao passar do tempo.

Os softwares não estão livres de sofrer mudanças. A manutenção e a evolução de sua estrutura são essenciais para retardar o seu desgaste e envelhecimento. A importância de se adotar essa técnica é tornar cada vez mais

longa a vida de um software sem que ele perca sua qualidade e suas funcionalidades, tornando assim mais seguro à futuras mudanças.

Com o passar do tempo os softwares vão envelhecendo e sua estrutura já não atende mais as atuais demandas. Isso pode levar ao fim da vida de um software, a não ser que o mesmo esteja preparado para as mudanças, o que nem sempre é uma tarefa fácil. É preciso observar o comportamento do software para tentar ao menos retardar o seu processo de envelhecimento.

Diante dessa questão, tem-se a evolução de software a fim de determinar até quando um software pode ser útil e como fazer para que o software continue a atender às necessidades do mercado de trabalho sem perder sua qualidade e sua funcionalidade.

Este trabalho tem o objetivo de apresentar os conceitos de evolução de software, abordando também a manutenção de software. A pesquisa tem como objetivo inicial caracterizar e conceituar a evolução de software, bem como mostrar as tarefas de manutenção, engenharia reversa e reengenharia e efeitos colaterais que a manutenção pode causar.

Um outro objetivo é abordar conceitos e técnicas da evolução, seu processo de envelhecimento, as mudanças que ocorrem com o passar do tempo, o nível de qualidade em relação à evolução, as dificuldades em evoluir o software.

Este trabalho tem como objetivo mostrar a importância de se adotar as técnicas de evolução de software e os pontos importantes que tornam o processo de evolução mais eficaz. Entender as causas do envelhecimento do software, e que atitudes devem ser tomadas para retardar esse processo.

Este trabalho está organizado como a seguir. O capítulo 2 apresenta a caracterização, a importância de evoluir um software, o porquê de adotar a técnica de evolução, as características da manutenção com base na evolução de software e as perdas e o ganho que podem ocorrer em um processo de evolução de software.

O capítulo 3 trata especificamente das metodologias de evolução, apresenta as leis de Lehman que servem como base para evolução de softwares, aborda manutenção evolutiva e técnicas de engenharia reversa e reegenharia.

No capítulo 4, é apresentada a aplicação de métodos que podem ser utilizadas na evolução de software como a gerência de configuração de software, além das ferramentas para auxiliar o controle da evolução de sistemas.

O capítulo 5 apresenta as considerações finais deste trabalho.

2 EVOLUÇÃO DE SOFTWARE

2.1 INTRODUÇÃO

Este capítulo mostra como os sistemas durante o passar dos tempos precisam evoluir para se adequar as mudanças na tecnologia e nos negócios. Também é caracterizada a evolução de software através de vários conceitos, identifica-se a importância de evoluir um software e porque adotar essa técnica. São mostradas algumas características da manutenção de software com base na evolução e como este processo pode influenciar de forma positiva e/ou negativa nas etapas evolutivas de um software.

2.2 HISTÓRICO

No início da era da computação softwares não eram produzidos em grande escala, dessa forma o controle de sua produção era mais simples de ser feito, não existiam métodos para controlar o desenvolvimento e nem equipes para realizar um controle da produção. Não existiam muitos métodos sistemáticos para realizar a programação, sua utilização ficava para segundo plano. O desenvolvimento de software era feito sem administração de forma virtual, ou seja, sem planejamento (PRESSMAN, 2006).

Este método de produção influenciava no prazo de entrega e nos custos de produção do software. Os softwares também eram influenciados pelo hardware que possuía alto custo e baixo poder de processamento e de armazenamento de informações (LOCATELLI ; KOMOSINSKI, 2003).

Segundo Pressman (2006), durante as três primeiras décadas da era do computador o maior desafio era desenvolver um tipo de hardware com custo de processamento e armazenagem de dados baixo. Foi o que ocorreu ao longo da década de 1980, onde os avanços na microeletônica possibilitaram aos computadores um custo cada vez mais baixo e um alto poder de processamento.

Para melhor entender a evolução de software é importante analisar os caminhos tomados na produção do software durante cinco décadas. A ilustração a seguir representa a evolução na produção durante cinco décadas e as diferentes questões norteadas de sua produção.

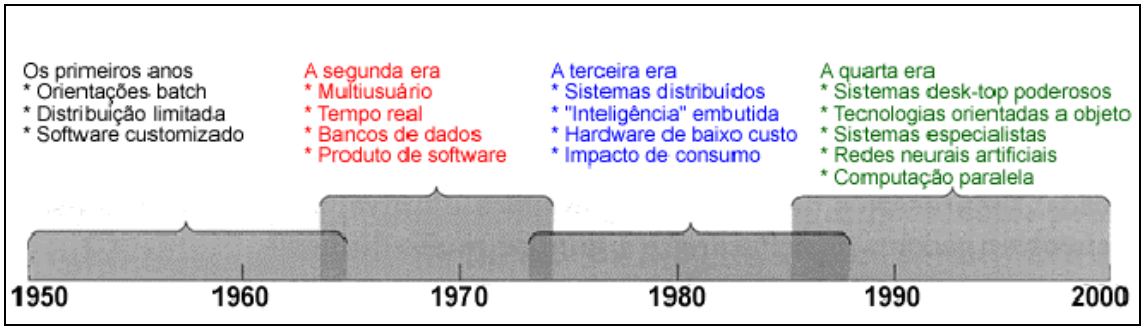

ILUSTRAÇÃO 1 – A evolução do software
Fonte: PRESSMAN, 2006.

De 1950 a 1960 os softwares eram produzidos sobre medida para cada aplicação através de orientações batch, dessa forma, a distribuição dos softwares eram prejudicadas. Praticamente todos os projetos de softwares produzidos ficavam dependentes de uma única pessoa, não havia quase documentação alguma, assim as empresas dependiam muito de seus funcionários (programadores). O software era desenvolvido pela propia pessoa ou organização que iria utilizá-lo. Suas manutenções eram feitas pela mesma pessoa que o desenvolvia, já que a rotatividade de empregos era baixa, assim os defeitos eram corrigidos por quem desenvolveu o sistema (PRESSMAN, 2006).

Já entre 1960 e 1970 ocorrem mudanças importantes no desenvolvimento de software, os sistemas começam a possuir interatividade através da utilização da multiprogramação e sistemas multiusuários; surgem os sistemas de tempo real que processavam instruções com maior velocidade gerando saídas rápidas, o que levou ao surgimento de sistemas que gerenciam bancos de dados. Começaram a ser criadas *softwares houses,* onde o desenvolvimento dos sistemas eram feitos em larga escala para milhares de clientes, porém isso levou a um problema, a manutenção do software, como administrar o atendimento personalizado a tantos clientes? Como detectar rapidamente as falhas e modificá-las? Por esses problemas surgiu a Crise do Software. Segundo Pressman (2006), surgira um grande problema, todos os programas deveriam ser corrigidos ao serem detectadas falhas, e teriam

que ser alterados para adaptarem as novas tecnologias e às exigências dos usuários. Assim foi criada uma nova atividade conhecida como “manutenção de software”, a qual começou a absorver muitos recursos.

De 1970 a 1980 os computadores pessoais começam a surgir, o uso de microprocessadores e estações de trabalho aumentam a produção de bens e serviços, as empresas que produzem software tem um alto crescimento e o software começa a se diferenciar em questão do hardware. O computador estava se tornando mais acessível, e o software adquiria mais qualidade (PRESSMAN, 2006).

De 1980 a 1990 surge a quarta era do software caracterizada por sistemas distribuídos, inteligência embutida, hardware de baixo custo e consumo. Os paradigmas de programação começam a mudar e programadores começam a utilizar conceitos de programação Orientada a Objetos (PRESSMAN, 2006).

Nos dias de hoje permanecem os sistemas desktop poderosos, tecnologias orientadas a objetos, sistemas especialistas, redes neurais artificiais e computação paralela. No início da década de 1980, a revista Business Week apresentou uma manchete que dizia: Software: Uma Nova Força Propulsora. Era dito que o software se transformara e tomado com mais atenção pela parte administrativa. Já em meados de 1980, surgira uma reportagem por Fortune que lamentava uma crescente defasagem de software. Ao final de 1980, com intuito de alertar os gerentes, a Business Week lança uma nova matéria com o título: Armadilha do Software – automatizar ou Não. No início da década de 1990, a Newsweek perguntava: Podemos confiar em nosso Software? Enquanto isso o Wall Street Journal falava sobre as dificuldades de uma grande empresa de software em um artigo que dizia: Criar Software Novo: Era uma tarefa Agonizante... Através dessas manchetes, o software é apresentado de uma nova forma, mostrando sua importância, as oportunidades e os perigos que ele apresenta (PRESSMAN, 2006).

As mudanças estão sempre ocorrendo enquanto novas idéias surgem e o ambiente de trabalho se modifica. É extremamente importante estar preparado para novas etapas a serem cumpridas. A evolução de um software é necessária para realizar novas tarefas e alcançar novos objetivos. É preciso estar atento e observar as constantes mudanças que possam vir a ocorrer em um ambiente de funcionamento de um software, pois são essas mudanças que ocasionam a modificação de um software. Preparar um software para tais mudanças pode aumentar seu tempo de vida em ambientes que sofrem constantes modificações.

2.3 CARACTERIZAÇÃO

O ambiente atual é extremamente mutável, sujeito as constantes transformações. Cada mudança traz um novo desafio, o de se adaptar e dessa forma, procurar atender aos novos padrões estabelecidos por essas inovações.

As mudanças nos meios de trabalho afetam não apenas as pessoas, mas também os softwares. Para que um software continue a cumprir suas tarefas ele deve evoluir junto com o ambiente onde é utilizado.

A seguir são apresentadas algumas classificações feitas sobre o processo de evolução de software. A princípio, Araújo e Travassos (2004) define a evolução de Software como o "...exame do comportamento dinâmico das características dos sistemas, como elas mudam ao longo do tempo".

Com outras palavras, classifica-se o processo de evolução de software como sendo o comportamento dinâmico dos sistemas, como eles são mantidos e expandidos ao longo de seu ciclo de vida (KEMERER e SLAUGHTER, 1999). A dinâmica de evolução de programas é o estudo dos processos de mudança do sistema (ARAÚJO, 2005).

Por último, a evolução de software se ocupa de modificar os sistemas de software existentes, para que eles atendam a novos requisitos (BONACIN, 2006).

A partir dessas definições pode-se dizer que evolução é uma técnica pela qual se observa o comportamento dinâmico do software durante um período de tempo, inspecionando suas mudanças que visam a atender novos requisitos.

2.4 A IMPORTÂNCIA DA EVOLUÇÃO DE UM SOFTWARE

Muitos softwares que são desenvolvidos não possuem sua estrutura maleável para futuras modificações. Isso ocorre pela falta de conhecimento das técnicas de desenvolvimento de software ou pela não utilização das mesmas.

Na maioria das vezes os sistemas são desenvolvidos para suprir determinada demanda imposta por clientes que necessitam o quanto antes do

software pronto para o uso. Ao término do desenvolvimento do software ele é implantado e assim utilizado pelos usuários durante um período de tempo, até o momento em que este software necessite de ser alterado para realizar novas tarefas. É neste momento, na fase da manutenção do software que é possível notar as irregularidades que o software possui e que não foram bem implementadas durante o processo de desenvolvimento.

A dependência gerada pelo software não permite que o mesmo seja descartado, assim só resta construir um novo software, o que provavelmente leva muito tempo e necessita alto investimento, ou reestruturar o software através de manutenções o que também pode ocasionar em altos custos. Para tentar minimizar estes problemas vem-se realizando estudos sobre as técnicas de evolução de software (ARAÚJO ; TRAVASSOS, 2004).

Os programas, assim como as pessoas, envelhecem. Não é possível deter o envelhecimento do software, porém é possível entender suas causas e assim tomar atitudes para evitar seus efeitos, reverter danos por algum tempo e assim estar preparado para o dia em que o software não será mais viável (ARAÚJO ; TRAVASSOS, 2004).

Por mais correto que seja o desenvolvimento de um software, este não está livre de manutenções. Segundo Leite (2001), o processo de construção de software é cada dia mais baseado no conceito de evolução, ou seja, normalmente se modifica algum software já existente. Com as técnicas de evolução é possível observar como o software está reagindo a modificações para tentar simplificar as próximas alterações.

Com base nas leis de Lehman, que são tratadas no capítulo 3, é possível identificar mudanças que um software pode estar sujeito a sofrer. De acordo com Araújo e Travassos (2004), através das leis e da classificação que Lehman impõe sobre os sistemas, provê-se um vocabulário amplamente aceito para discutir a natureza das mudanças dos softwares. Por essas idéias é possível projetar sistemas para serem mais flexíveis, realizar o planejamento das manutenções, além de entender e controlar de uma melhor forma o desenvolvimento do software, e não apenas reagir aos problemas que ocorrem (PFLEEGER apud ARAÚJO ; TRAVASSOS, 2004).

2.5 POR QUE ADOTAR A TÉCNICA DE EVOLUÇÃO?

Com o passar do tempo os softwares vão “envelhecendo”, sua estrutura se torna ultrapassada e por já não satisfazerem suas tarefas como antes, são descartados. Segundo Bonancin (2006), o software é inerentemente flexível e pode ser alterado. Raramente é necessário descartar um software inteiro por ele não ser capaz de realizar determinada tarefa ou atingir um novo resultado. Em determinadas circunstâncias é preferível adaptar o software ao invés de eliminá-lo totalmente. Porém para que isso seja possível é necessário que sejam adotadas técnicas de evolução de software, dessa forma o processo de reestruturação de um software pode ser possível evitando dessa forma que o mesmo perca sua utilidade.

Segundo Pfleeger apud Araújo e Travassos (2004), é necessário antecipar os caminhos em que o software sofre mudanças, dessa forma pode-se modificá-lo mais facilmente para acomodar tais necessidades. Porém, antecipar-se às mudanças não é uma tarefa fácil, uma vez que existem muitas razões pelas quais os sistemas mudam. Uma delas é relatada por Bonancin (2006) que comenta que as alterações dos requisitos ocorrem em função das mudanças do negócio, dessa forma o software que dá suporte ao negócio também deve ser alterado.

Adotar técnicas de evolução de software pode ser de grande importância em meio às transformações que o software está sujeito a sofrer. Segundo Lehman apud Araújo e Travassos (2004), as Leis de Evolução de Software descrevem como um sistema se comporta ao longo de sucessivas versões. Dessa forma as futuras mudanças são de certa forma, controladas. Como as modificações estão presentes na vida de um software, é preciso nos preparar para as próximas e tentar reverter através das técnicas de evolução de software, o envelhecimento do software, ou seja, adiar o dia em que este já não poderá ser mais viável.

2.6 REFLEXÃO DA MANUTENÇÃO SOBRE A EVOLUÇÃO DE SOFTWARE

Ao falar de Evolução de Software, não se pode esquecer que junto a este processo está presente a manutenção do software. Para evoluir um software é

necessário modificá-lo, ou seja, realizar algum tipo de manutenção seja em sua estrutura ou documentação.

Os processos de evolução e manutenção de software possuem uma forte ligação e devem ser distinguidos por possuírem diferentes características. A manutenção de software se refere às atividades que ocorrem em qualquer época após ser implementado um novo projeto de desenvolvimento de software, enquanto que a evolução de software é o exame do comportamento dinâmico das características dos sistemas, como eles mudam ao longo do tempo (ARAÚJO ; TRAVASSOS, 2004).

Nota-se que o processo de Evolução de Software é baseado na teoria das observações feitas sobre cada modificação de um sistema e como este se comporta. Já o processo de manutenção de software, se caracteriza por ser um processo mais “prático” que pode utilizar as teorias e observações feitas durante um processo de evolução para buscar melhores formas de realizar a manutenção sem prejudicar as funcionalidades e estruturas do sistema.

Com o objetivo de caracterizar melhor a atividade de manutenção de software, é possível dizer que manutenção é o processo de modificar um componente ou um sistema de software, após sua entrega, a fim de corrigir falhas, melhorar seu desempenho ou outros atributos, ou mesmo adaptá-lo a mudanças ocorridas no ambiente. A manutenção de software começa a partir de uma solicitação de *release* do usuário, ou seja, no momento em que o cliente necessite que alterações sejam feitas para corrigir o software ou mesmo acrescentar novas funções a ele. Os usuários por sua vez, não possuem conhecimento técnico, e por este motivo acabam levando em conta que a tarefa de alteração de um software seja simples de se fazer, o que na realidade é bem mais complicado do que se pensa. Lógico que essas modificações devem ser feitas de acordo com um contrato aceito por ambas as partes (MURTA, 2007).

A manutenção é inevitável, uma das causas que levam a manutenção são os requisitos do sistema que mudam devido as mudanças no ambiente. Para que um sistema continue a ser útil em um ambiente, ele tem que ser mantido atualizado conforme as novas regras. As manutenções são realizadas para reparar falhas, adaptar e adicionar ou modificar a funcionalidade dos sistemas para satisfazer novos requisitos (ARAÚJO, 2005).

Para organizar melhor esse processo, a manutenção pode ser dividida em diversos tipos. Segundo Murta (2007) os tipos de manutenção podem ser classificados em manutenção emergencial, corretiva, preventiva, adaptativa, e perfectiva, onde as três primeiras são etapas consideradas como correção do software e as duas últimas fazem parte do processo de evolução do software. São abordadas as manutenções adaptativa e perfectiva que se encaixam na etapa de evolução de software.

Ao falar de manutenção adaptativa, diz-se que esta é a atividade de manter o software usável após mudanças no ambiente, já a manutenção perfectiva provê melhorias para o usuário através da melhora dos atributos de qualidade de software (MURTA, 2007).

A manutenção adaptativa procura adaptar o sistema a mudanças nas necessidades do usuário ou do ambiente, onde essas atividades são originadas por algum tipo de solicitação de melhoramento. Dentro da etapa de manutenção adaptativa estão incluídas as atividades como adição de funcionalidades, mudanças no formato dos dados de entrada e adaptação a novas regras de negócio do usuário. Para realizar tais adaptações no sistema, é necessário identificar as partes da arquitetura que são envolvidas, elaborar alternativas para sua alteração avaliando essas alternativas e finalmente implementar a alternativa selecionada. Porém para fazer tais mudanças é preciso estar preparado, estudar as formas e o comportamento de evolução do software pode auxiliar neste processo (SOUZA, 2005).

A importância de se estudar as etapas evolutivas do software não está somente na parte da adaptabilidade, mas também nas fases de manutenções perfectiva. A parte de manutenções perfectivas tende a modificar um sistema de modo a aumentar a qualidade do software ou de sua documentação, sem modificar sua funcionalidade. Nesta categoria, podem ser identificados os esforços para aumentar a legibilidade do software, para melhorar sua performance, para incrementar a reusabilidade, entre outros. Nesta etapa, é preciso identificar os aspectos de qualidade no qual se pretende realizar melhorias, ao identificar esses pontos deve-se tratá-los como defeitos e aplicar devidas correções (SOUZA, 2005).

Em ambos processos de manutenção é possível através do estudo da gerência de configuração de software, controlar a evolução dos sistemas de software. O controle de modificação e versão apóia o processo de evolução de um

sistema, além disso, pode-se utilizar ferramentas de controle de versão e modificação para tornar o processo mais simples de ser gerenciado. Ferramentas como o Subversion, CVS, Star Team podem ser utilizadas para o controle de versões, já as ferramentas Bugzzila, Mantis e ClearQuest, por exemplo, podem ser utilizadas para o controle de modificações (MURTA, 2007).

Através de ferramentas e metodologias adequadas é possível tratar o processo de evolução de software com mais clareza. Buscar entender cada modificação feita em um sistema pode ajudar a simplificar futuras alterações. Às vezes é possível atingir resultados esperados na manutenção, porém os estudos da evolução de software proporcionam alternativas que podem controlar melhor o processo de manutenção a fim de alcançar melhores objetivos e aumentar o grau de manutenibilidade do software.

2.7 PERDAS E GANHOS NO PROCESSO DE EVOLUÇÃO

Dizer qual a melhor forma para obter ganhos e não haver perdas em um processo de evolução de software é bem relativo já que cada software possui uma estrutura diferente e atende para diferentes tipos de ambientes. Porém algumas atitudes podem ser tomadas para obter melhores resultados no processo de evolução de software.

Observar como o software se comporta ao longo de suas alterações pode ser um meio de reverter perdas, pois dessa forma é possível ter um controle maior das próximas modificações que forem realizadas no sistema. Estar preparado para algum imprevisto que possa ocorrer no processo de manutenção do software é importante, pois pode amenizar possíveis danos neste processo.

Um ponto forte que auxilia no processo de evolução são as leis de Evolução de Software propostas por Lehman, onde através delas é possível identificar as características do software e dizer se este está próximo ou não da necessidade de sofrer algum tipo de modificação (ARAÚJO ; TRAVASSOS, 2004).

De acordo com Torquato (2006), estruturar um software para sua evolução não é tarefa fácil, é necessária a experiência dos projetistas para mensurar de forma adequada o grau de evolução que se deseja em determinado momento. É

importante realizar uma documentação adequada onde todos os envolvidos na equipe tenham uma clara importância da documentação.

É preciso levar em consideração também a importância da equipe de desenvolvimento, a experiência que cada um possui e o grau de rotatividade dos integrantes da equipe de desenvolvimento. Um integrante que possui um vasto conhecimento da estruturação e desenvolvimento daquele software ao abandonar a equipe em meio a um processo de alteração do sistema pode tornar o processo a ser mais demorado, desestruturando de certa forma o modo de que o software vinha sendo alterado e assim gerando possíveis perdas na qualidade do sistema, caso não seja alocado outra pessoa com devidos conhecimentos para dar continuidade a tal tarefa (BONACIN, 2006).

Segundo Bonacin (2006), a evolução de um sistema assim como a evolução do software é inerentemente dispendiosa por algumas razões, tais como as mudanças que são propostas necessitam ser analisadas cuidadosamente tanto sob o ponto de vista técnico como de acordo com uma perspectiva de negócios, ou seja, o software deve procurar suprir as tarefas impostas pelo usuário além de realizá-las sem apresentar falhas e impedimentos na realização do trabalho do cliente.

Um outro ponto que deve ser considerado é a necessidade de manter em funcionamento e atualizados os sistemas legados, que são sistemas antigos, porém importantes que continuem funcionando em uma empresa (BONACIN, 2006). Dependendo de como esses sistemas foram projetados sua evolução pode ser trabalhosa, porém necessária já que a dependência causada por esse sistema é grande. A evolução de um software também pode ocasionar na expansibilidade do sistema e no aumento da sua complexidade o que pode levar a um aumento na sua taxa de falhas (MARTINS, 2001).

O processo de evoluir um software pode ser muito útil para tentar adiar o fim do uso de um software, ou seja, buscar o seu rejuvenescimento, porém este processo deve ser bem planejado para que contrapartidas, como por exemplo, a perda da qualidade, aumento da complexidade, crescimento da taxa de falhas não venham a ocorrer (MARTINS, 2001). A ilustração 2 mostra como as modificações aumentam a complexidade do software e o seu número de falhas.

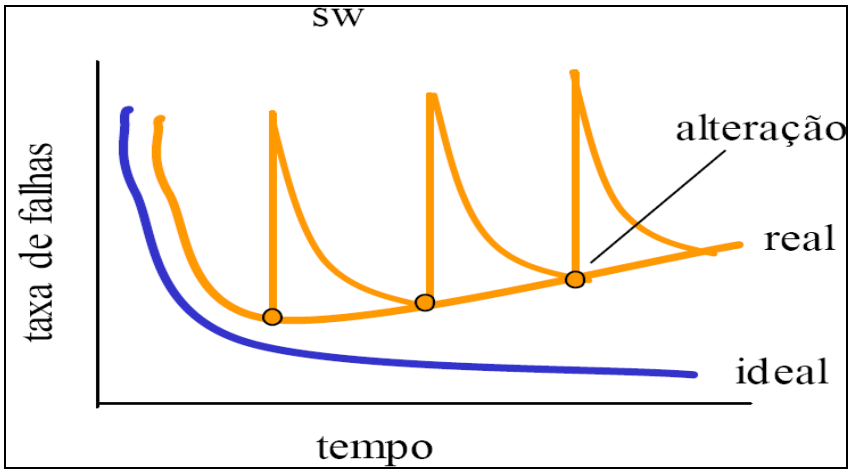

ILUSTRAÇÃO 2 – Taxa de Falhas X Tempo
Fonte: MARTINS, 2001.

2.8 CONCLUSÃO

A atividade de evolução de software é considerada uma nova metodologia para o estudo do comportamento dos sistemas, a fim de entender como estes reagem para estipular medidas a serem aplicadas com intenção de estender por mais tempo a atividade que este realiza em determinado ambiente de negócios.

Com as rápidas transformações no ambiente em que um software está aplicado, a necessidade de estar preparado para cumprir novas tarefas é extremamente importante, uma vez que o não cumprimento dessas tarefas pode levar ao descarte do software. Entender como os softwares sofrem mudanças é importante a fim de facilitar futuras mudanças em sua estrutura. Evoluir é necessário, mas pode demandar tempo e custo, o que muitas vezes pode causar uma barreira para aplicar este processo. A evolução de software não é uma tarefa simples, mas possível de ser realizada.

3 METODOLOGIAS DE EVOLUÇÃO

3.1 INTRODUÇÃO

Neste capítulo são apresentadas as oito Leis de Lehman utilizadas na Evolução de Software como base para evoluir softwares. Também são mostradas as formas de manutenção evolutiva que um determinado processo de evolução de software pode aplicar na evolução de sistemas, levando em consideração as diferenças dos processos.

Além disso, ainda são demonstradas as técnicas de engenharia reversa e reengenharia de software, as quais são utilizadas com o intuito de esclarecer melhor o funcionamento de um software a fim de poder alterá-lo e envolvê-lo em um ambiente de evolução de software.

3.2 LEIS DE LEHMAN

No estudo da Evolução de Software, Lehman, é considerado um dos precursores da área de evolução de software. Foi ele quem criou as oito Leis de Lehman que podem ser consideradas como base para uma teoria de evolução de software.

As Leis de Lehman começaram a ser formuladas no início dos anos 70 através da análise do processo de programação da IBM. Neste período foram formuladas as três primeiras leis, na década seguinte foram apresentadas mais duas leis. Na década de 90 foram apresentadas mais três leis, formando assim as oito leis de Lehman (CHRISTOPH, 2004). Apesar das leis terem sido elaboradas há décadas atrás, não significa que sejam inaplicáveis aos sistemas de hoje, pelo contrário. As novas análises não contradizem as leis de evolução de software, mesmo pelas leis terem sido elaboradas na década de 70, ou seja, são ainda relevantes para analisar as medidas de evolução atualmente (ARAÚJO ; TRAVASSOS, 2004).

Os sistemas podem ser classificados em três tipos. Segundo Lehman apud Araújo e Travassos (2004) cada sistema possui uma forma diferente de evolução de acordo com sua natureza:

- S-Systems (*specification*): neste tipo de sistema o problema é bem definido e a solução é bem conhecida. Está relacionado diretamente com o mundo real e, se este muda, o resultado é um problema completamente novo que deve ser especificado assim, são improváveis de mudar. Um exemplo que se enquadra neste tipo de classificação são os sistemas para operações em matrizes;
- P-Systems (*problem*): mais dinâmico que o S-Systems, baseia-se em uma abstração prática do problema, ao invés de uma especificação completamente definida. Está sujeito a mudanças incrementais, a solução produz informação que é comparada com o problema e depende em partes do analista que gerou os requisitos. Os sistemas para jogo de xadrez são exemplos de sistemas desse tipo;
- E-Systems (*embedded*): muda de acordo com as mudanças naturais do mundo real, já que está embutido no mundo real. Como o sistema é uma parte do mundo modelado, são provavelmente submetidas às mudanças quase constantes. Como exemplos, podem ser citados os sistemas para a área financeira.

Cada sistema possui natureza e comportamentos diferentes ao evoluir em ambientes desiguais em que operam, mesmo com essas diferenças as leis de Lehman podem ser aplicadas em variadas situações de evolução de software. Essas leis descrevem um sistema em termos dos caminhos em que se relaciona com o ambiente em que se opera. A seguir são apresentadas as oito leis de Lehman e algumas de suas características (LEHMAN apud ARAÚJO ; TRAVASSOS, 2004).

Sobre a mudança contínua, primeira lei de Lehman criada em 1974. A respeito desta lei é possível dizer que:

- um produto em uso ou está em mudança constante ou se torna progressivamente menos útil;
- sistemas devem ser continuamente adaptados senão tornar-se-ão progressivamente menos satisfatórios;

- o processo de decaimento continua até que seja mais barato substituir o sistema com uma versão recriada.

Esta lei sugere que os sistemas podem sofrer um processo parecido com o envelhecimento humano, que pode resultar em inconsistências do software e do domínio em que este está inserido, já que este domínio faz parte do mundo real e está em contínua evolução. A evolução deve ser feita sobre os retornos dos usuários levando em consideração o seu nível de satisfação, dependendo do nível de resistência em se evoluir o software ou mesmo de adaptá-lo as necessidades do usuário, seu nível de satisfação cai com o tempo (CHRISTOPH, 2004).

Sobre a complexidade crescente ou incremento da complexidade, segunda lei de Lehman criada em 1974, pode-se dizer que:

- a mudança constante continuamente introduz complexidade no produto, deteriorando a estrutura do mesmo;
- se não for desenvolvida nenhuma atividade explícita do controle da complexidade, a manutenção do produto deixa de ser possível e este torna-se inútil.

À medida que a necessidade de adaptação de um software aumenta e as mudanças são sucessivamente implementadas, interações e dependências entre os elementos dos sistemas crescem de forma desestruturada a levar um crescimento da entropia do sistema. Quanto mais mudanças forem feitas no sistema, sua estrutura original vai se tornando mais fragmentada e o custo de mais mudanças aumenta gradativamente até o ponto em que estes custos não são mais viáveis. Dessa forma é necessário realizar a reestruturação do software para tentar diminuir sua complexidade (CHRISTOPH, 2004).

A auto-regulação, terceira lei de Lehman criada em 1974, afirma-se que:

- é a lei fundamental da evolução do produto;
- o processo de evolução do produto é uma dinâmica auto regulada com tendências estatísticas determináveis e invariâncias;
- sistemas de software exibem comportamentos regulares e tendências que podem ser mediadas e previstas.

A evolução de software é implementada por um grupo de técnicos, que opera dentro de uma organização maior. Os interesses desta organização e seus objetivos se estendem bem acima do sistema em questão. Para garantir que as

normas operacionais são seguidas e os objetivos organizacionais são atingidos em todos os níveis, são estabelecidos pontos de controle pela gerência. Os pontos de controles são exemplos de mecanismos de estabilização, podendo gerar controles positivos e negativos. Existem vários outros, e juntos eles estabelecem uma dinâmica disciplinada cujos parâmetros são, pelo menos em parte, normalmente distribuídos. Ao longo do tempo este grupo estabelece uma dinâmica que fará com que o esforço incremental gasto em cada nova versão permaneça constante durante a vida do sistema (CHRISTOPH, 2004).

Sobre a conservação da estabilidade organizacional, quarta lei de Lehman estipulada em 1978, pode-se dizer que:

- a taxa de atividade global em um produto em evolução é estatisticamente invariante ao longo de seu ciclo de vida;
- atributos organizacionais, como produtividade, não exibem grandes flutuações;
- recursos e resultados alcançam um nível ótimo, e adicionar mais recursos não muda significativamente os resultados.

Entre todas as oito leis de evolução de software, esta é sem dúvida a menos intuitiva, pois ainda se acredita que a taxa de atividade global gasta em um sistema em evolução é decidido pelos gerentes responsáveis. Projetos analisados mostram o contrário, verificando que essa taxa se estabiliza em um nível constante, e que na prática o nível de atividade de um projeto não é decidido exclusivamente pela gerência. Isso ocorre, pois o nível de atividade vai ser decidido pelas necessidades dos usuários e seus retornos, como mostrado na terceira lei de Lehman. Este nível após um período de tempo, tende a se manter constante, sendo que o nível de pessoas no software não pode crescer indefinidamente, pois isso vai acarretar em um aumento igualmente grande na entropia do sistema, podendo resultar em uma diminuição da taxa de atividade global (CHRISTOPH, 2004).

A conservação da familiaridade é a quinta lei de Lehman estipulada em 1978. A respeito desta lei é possível dizer que:

- durante o ciclo de vida de um produto o conteúdo de cada versão é estatisticamente invariante;
- o conteúdo de sucessivas versões de programas em evolução (mudanças, adições, exclusões) é estatisticamente invariante;

- após um tempo, o efeito de versões de manutenções sucessivas faz pouca diferença na funcionalidade geral.

Um fator determinante na evolução de um software é a familiaridade de todos os membros da equipe com os objetivos desta, quanto mais mudanças forem necessárias, maior vai ser a dificuldade de que toda a equipe esteja ciente dos objetivos. A qualidade e taxa de progresso, além de outros parâmetros são influenciados, até mesmo limitados, pela taxa de aquisição da informação necessária pelos participantes coletivamente e individualmente. Através de dados coletados pode-se dizer que esta relação não é linear, porém existem um ou mais tamanhos críticos, que se excedidos acarretam em mudanças comportamentais (CHRISTOPH, 2004).

Sobre o crescimento contínuo, sexta lei de Lehman estipulada em 1978, é possível afirmar que: O conteúdo funcional de um sistema deve ser continuamente incrementado para manter a satisfação do usuário ao longo do ciclo de vida.

Após o lançamento de um software, mudanças são necessárias para garantir a satisfação do usuário, estas mudanças podem ser correções de erro, adições de novas funcionalidades ou melhorias em funções pré-existentes. Muitas destas mudanças quase não são planejadas pela equipe de desenvolvimento na época da primeira versão, ou foram causadas devido a mudanças no ambiente em que o software está inserido (podendo invalidar assim algumas suposições feitas anteriormente). Mudanças que não são planejadas inicialmente geram necessidades de criar aplicações externas e módulos extras para o software, o que causa um inevitável aumento do conteúdo funcional deste programa (CHRISTOPH, 2004).

O declínio da qualidade ou qualidade decrescente, sétima lei de Lehman foi estipulada em 1994. A respeito desta lei é possível considerar: A qualidade de um sistema entrará em declínio a menos que seja rigorosamente mantida e adaptada às mudanças do ambiente operacional.

O fato de um software ser criado com um número de recursos e tempo limitados, e por estar inserido em um domínio suscetível a efeitos externos, causa uma determinada imprevisibilidade a este software. Mesmo que um software funcione de forma satisfatória por muitos anos, não significa que ele vá continuar funcionando da mesma forma nos anos seguintes.

A sétima lei de Lehman diz que o nível de incerteza de um software aumenta com o tempo, a não ser que seja feito um esforço para detectar e corrigir

as causas desta incerteza, sendo que este esforço evolutivo deve ser contínuo para todas as novas versões do software. Esta lei também se refere ao critério de satisfação da comunidade, que conforme passar do tempo ficam mais exigentes com o software que utilizam, aumentando assim o critério de satisfação. A concorrência surge no mercado com novos produtos, novas tecnologias são criadas, novas funcionalidades passam a ser necessárias, dessa forma o software que tinha uma qualidade satisfatória anos atrás não necessariamente tem a mesma qualidade anos depois (CHRISTOPH, 2004).

O sistema de realimentação ou sistema de retorno é a oitava lei de Lehman criada em 1972 e modificada em 1996. A respeito desta lei pode-se considerar: O processo de evolução de um sistema constitui em realimentação em multi-nível, multi-interação e multi-agente do sistema e deve ser tratado de forma alcançar significativas melhorias.

Esta lei foi formulada nos anos 70, porém somente foi apresentada na década de 90. Os estudos mostravam que o sistema de evolução de um software constitui um sistema complexo que é constantemente realimentado por retornos de seus usuários. A vida de um software pode ser considerada como um ciclo bem definido de retornos positivos e negativos ao longo de cada versão do software. As quantidades de retornos negativos e positivos vão estipular ao longo prazo, a taxa de crescimento do sistema a qual pode ser controlada por fatores como quantidade de verba, números de usuários solicitando uma nova funcionalidade ou reportando algum erro, interesses administrativos e tempo entre uma versão e outra. Caso não se consiga evoluir um software, este é descartado no futuro, já que seu envelhecimento é inevitável ao longo do tempo. Com isso o software não satisfaz mais as necessidades dos usuários e é, portanto esquecido e substituído por outro mais atual que atenda às novas necessidades (CHRISTOPH, 2004).

Percebe-se através das leis de Lehman que, a evolução do software está intimamente ligada ao usuário. Segundo Christoph (2004) a oitava lei de Lehman mostra claramente o papel dos usuários na evolução do software. Pode-se afirmar ainda que todas as leis de Lehman refletem a relação direta existente entre o nível de satisfação de usuários finais e os requisitos de evolução.

Com base nos estudos de Christoph sobre a aplicação das leis de evolução em software livre, coloca-se em questão uma citação feita por Christoph (2004), a qual diz que foi examinada a evolução de software livre baseando-se nas

leis de evolução de software propostas por Lehman. Esperava-se que todas as leis se aplicassem no processo, já que a evolução de um software livre é muito mais informal do que um processo estruturado de grandes empresas, assim os estudos apresentados por Godfrey com o Linux mostram que raramente é o que ocorre.

Com relação a este estudo, é dito que as leis podem não ser aplicáveis ao processo de evolução de software livre, isto pela forma com que os softwares livres realizam seu processo de desenvolvimento, sendo este, processos que não são baseados nos padrões focados por Lehman, mas sim em processos semelhantes no que se conhece hoje por *Extreme Programming.* Porém as leis não estão erradas, dessa forma não é possível dizer que as leis não se aplicam ao desenvolvimento de softwares livres, elas apenas sugerem mais pesquisas.

Através das leis de Lehman apresentadas, entende-se de forma mais clara como ocorre o processo evolutivo de um software, identificar suas fases e necessidades de possíveis mudanças.

3.3 MANUTENÇÃO EVOLUTIVA

Em um processo de evolução de software é necessário que as atividades sejam voltadas para a manutenção do software para que as devidas alterações sejam implementadas e estruturadas no sistema, a fim de fazer com que o mesmo atenda às novas funções. A manutenção de software segundo IEEE apud Murta (2007), é classificada como sendo o processo de modificar um software, após a entrega, a fim de corrigir suas falhas, melhorar seu desempenho e adaptá-lo a mudanças ocorridas no ambiente. O processo de manutenção inicia-se logo após a construção de um software, no momento em que são feitas solicitações de mudança no software, podendo estas, ser freqüentes (MURTA, 2007).

Um processo de manutenção pode ser voltado para correções, onde este pode ser classificado em emergencial, corretiva e preventiva, e também pode ser voltado para a manutenção evolutiva, que pode ser classificada em adaptativa e perfectiva (MURTA, 2007). Este tópico tem seu estudo focado na análise de manutenção evolutiva.

Como foi dito a manutenção pode ser voltada tanto para correção quanto para evolução. A manutenção adaptativa tem como objetivo, adaptar o sistema a mudanças nas necessidades do usuário ou do ambiente. Essas atividades originam-se através de solicitações de melhoramento. Nelas estão incluídas atividades como adição de funcionalidade, mudanças no formato dos dados de entrada e adaptação a novas regras do usuário. Já a manutenção perfectiva modifica o sistema de modo a aumentar a qualidade do software ou de sua documentação, sem modificar sua funcionalidade. Nesta categoria estão incluídos os esforços para aumentar a legibilidade do software, a fim de melhorar a performance, incrementar a reusabilidade, entre outros (SOUZA, 2005).

Após a solicitação de mudança em um software a atividade de manutenção deve começar. Para aplicar mudanças através de manutenção adaptativa deve-se identificar as partes da arquitetura envolvida, elaborar alternativas, avaliar essas alternativas e com isso implementar a alternativa escolhida. Caso seja necessário aplicar manutenção perfectiva, é preciso identificar os aspectos de qualidade que são candidatos a melhoria, tratar os candidatos identificados como defeitos e assim aplicar a técnica de manutenção corretiva. A manutenibilidade do software, que é caracterizada pela facilidade de modificar ou adaptar um software, deve ser controlada em meio a estes processos a fim de minimizar os custos de manutenção, e tornar as próximas mais fáceis de serem entendidas (SOUZA, 2005). A ilustração 3 mostra os custos que se têm ao se aplicar um dos três tipos de manutenção: corretiva, perfectiva ou adaptativa.

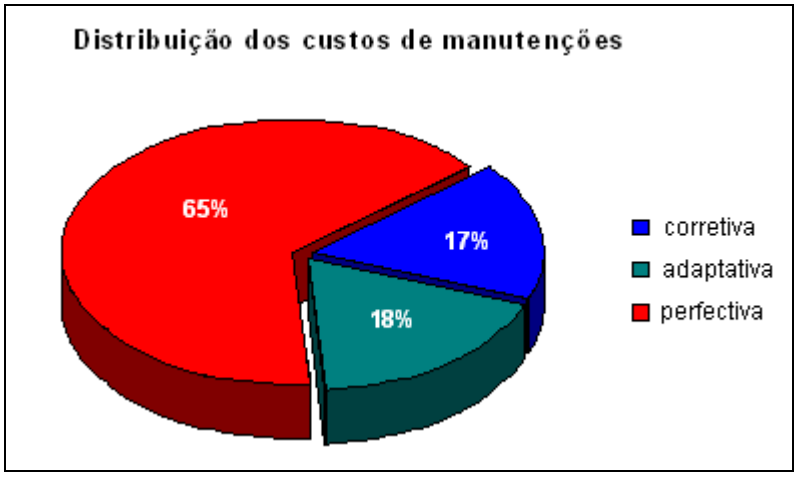

ILUSTRAÇÃO 3 – Distribuição dos custos por categoria
Fonte: MARTINS, 2001.

Percebe-se que na manutenção evolutiva, dentre as duas categorias tanto a manutenção adaptativa, quanto a perfectiva, executam processos sobre software já existentes e não sobre a construção de um novo software. Esses métodos são característicos de evolução de software, que busca melhorar um software já pronto e não descartá-lo e construí-lo todo novamente.

3.4 ENGENHARIA REVERSA

Hoje em dia, o processo de negócios está em constante mudança gerando assim novos objetivos a serem atingidos. Com isso, sistemas que atuam em ambientes desse tipo devem sofrer modificações a fim de acompanhar o processo de negócio. Segundo Saleh e Boujarwah apud Feltrim (1999), o mercado de software vem crescendo a cada dia e com ele as técnicas de desenvolvimento que na maioria das vezes são informais. Dessa forma a manutenção do software pode se tornar problemática já que a documentação associada a este software pode não estar de acordo com o código implementado. Além disso, as constantes modificações e o acréscimo de novas características ao software acarretam efeitos colaterais que podem não estar presentes na documentação.

Com documentações informais e incompletas, que não se referem ao software existente, tornam impossível o gerenciamento do processo de manutenção (SALEH ; BOUJARWAH apud FELTRIM, 1999). A facilidade de realizar a manutenção esta relacionada ao entendimento do software, a documentação que o sistema possui influencia neste entendimento, dependendo da qualidade da documentação o software é mais simples de ser interpretado (LEITÃO, 2001). Na maioria das vezes, a documentação é inexistente, incompleta e/ou desatualizada devido à fatores como o software ser muito antigo ou a falta de documentação de atualizações nas modificações do software, ou mesmo por desleixo da equipe de desenvolvimento. De acordo com Feltrim (1999), a Engenharia Reversa de Software, tem o propósito de recuperar as informações de projeto perdidas durante a fase de desenvolvimento, e de documentar o real estado do software a fim de auxiliar o processo de gerenciamento de manutenção.

A engenharia reversa pode ser útil em ambientes como estes onde é necessário compreender o software para evoluir e consequentemente adapta-lo às devidas mudanças.

Existem várias definições de Engenharia Reversa. Este processo pode ser classificado como o exame e compreensão do software, a fim de recapturar ou recriar seu projeto e identificar os requisitos atualmente implementados pelo sistema de forma a apresentá-los em um nível ou grau maior de abstração (BRAGA, 2006).

Já Feltrim (1999) define o processo de engenharia reversa como sendo um processo de investigação do software e não de mudança ou reprodução. De forma mais geral, é possível dizer que a Engenharia Reversa é uma atividade que trabalha com um produto existente tentando entender como o mesmo funciona e como é o seu comportamento em diversas circunstâncias (CANHOTA et al., 2005).

É identificado o propósito da engenharia reversa como a forma de entender o software com a finalidade de simplificar as atividades de expansão, correção, documentação, re-projeto ou re-programação em outra linguagem de programação. De forma sintetizada a Engenharia Reversa é o processo de voltar atrás no ciclo de desenvolvimento buscando entender como este foi elaborado. (FELTRIM, 1999).

Através desses conceitos entende-se de forma superficial o que é Engenharia Reversa, porém é preciso identificar o momento em que ela deve ser utilizada. Geralmente ela é utilizada quando é necessário trocar algum produto com defeito, ou mesmo, quando se quer conhecer como este funciona e não tem-se nenhuma documentação (CANHOTA JUNIOR, 2005). Para melhor entender as características de um software, este pode ser visualizado em diferentes níveis de abstração. Cada visualização abstrai características próprias da fase do ciclo de vida correspondente à abstração. Os níveis de abstração são mais altos nos estágios iniciais do ciclo de vida e mais baixos nos estágios finais. A representação em níveis mais altos facilita o entendimento de sistemas de software, ou seja, quanto mais alto for o grau de abstração de um software mais fácil é o entendimento de sua estrutura e funcionamento (BRAGA, 2006).

Dentro da Engenharia Reversa pode-se identificar elementos para serem utilizados ao aplicar esta técnica. Estes são classificados em quatro tipos, nível de abstração, conforme o nível de abstração aumenta, mais compreensíveis se tornam as informações, completitude do processo, o qual refere-se ao nível de detalhes que

é fornecido em cada nível de abstração, interatividade, reporta-se ao grau de participação do ser humano no processo de engenharia reversa, que de acordo com o aumento do nível de abstração a interatividade deve aumentar ou a completitude é prejudicada, direcionalidade, quando possui sentido único significa que toda a informação extraída do código fonte é usada durante as atividades de manutenção e quando possui sentido duplo a informação é utilizada para “alimentar” uma ferramenta de reengenharia (BRAGA, 2006).

Em meio a estes processos são utilizados documentos para realizar a engenharia reversa, esses documentos podem ser formados por informações dos usuários e/ou analistas, código fonte e documentações existentes, como por exemplo, manual de usuário, manual de sistema, DFDs e fluxogramas (BRAGA, 2006).

Nas manutenções adaptativas e evolutivas, as técnicas de engenharia reversa são utilizadas indiretamente, através do fornecimento de visões do software, para localizar os componentes onde são realizadas as mudanças e adições necessárias, e para auxiliar no controle da estrutura global do sistema modificado, através da produção de documentação. Segundo Braga (2006), os maiores benefícios de engenharia reversa são mais reconhecidos quando manutenções futuras tiverem como apoio a documentação produzida numa manutenção anterior.

Em relação ao reuso, que segundo Braga (2006) consiste em uma atividade que se destina a identificar software reutilizável, envolvendo a correta importação, reconfiguração e adaptação deste software para uma nova aplicação em um sistema de computação, a engenharia reversa mesmo não sendo focalizada na identificação e composição de componentes a partir de partes reutilizáveis, pode ser proveitosa em completar a documentação de novos sistemas compostos. De acordo com Leitão (2001) a engenharia reversa, é uma área importante, pois existe um grande volume de sistemas legados que precisam evoluir. O nome Engenharia Reversa é dito pelo fato de que esse processo parte do produto para sua definição, ao contrário da engenharia tradicional. Antes de evoluir um sistema de software é necessário revertê-lo de forma que este chegue a um grau de abstração mais alto para que haja uma melhor compreensão do que é o sistema, como funciona, e o que não funciona, para finalmente poder modificá-lo. A Engenharia Reversa é aplicada a três aspectos principais de um sistema, sendo estes, dados, processo e controle.

A partir da obtenção das informações necessárias para o entendimento do sistema, a engenharia reversa é aplicada e utilizada em sistemas que não possuem recursos necessários para evoluir de tal forma que sejam adaptados a novos computadores, software ou regras.

3.5 REENGENHARIA

Enquanto a Engenharia Reversa consiste em apenas analisar o sistema ou a ferramenta para criar uma representação dela, a Reengenharia vai além. Analisa-se o projeto, cria-se uma representação do mesmo e, através dessa representação, monta-se uma nova estrutura que funcione exatamente como a primeira, mas que não seja meramente uma cópia (CANHOTA JUNIOR, 2005).

A reengenharia pode ser definida segundo Chikofsky e Cross apud Braga (2006), como sendo o exame e a alteração de um sistema para reconstruí-lo de uma nova forma, seguida pela sua implementação. De acordo com Piekarski e Quináia (2000), o termo reengenharia está relacionado com a reconstrução de algo do mundo real, que tem como objetivo, independente de sua aplicação, construir algo com mais qualidade ou mesmo com a qualidade comparável ao produto inicial.

A reengenharia, também chamada de recuperação ou renovação, recupera informações de projeto de um software e utiliza essas informações para alterar ou reconstruir o sistema, preservando as funções existentes, ao mesmo tempo em que adiciona novas funções ao software, num esforço para melhorar sua qualidade global. Através destes conceitos é possível identificar que existe uma distinção entre reengenharia de software e o desenvolvimento de um novo software. Essa distinção está relacionada ao ponto de partida de cada um dos processos (PIEKARSKI ; QUINÁIA, 2000).

Enquanto o desenvolvimento de um novo software inicia-se com uma especificação escrita do software que é construído, a engenharia reversa inicia-se tomando como base um sistema já desenvolvido. Também é possível observar a diferença entre os objetivos da engenharia reversa e os da reengenharia. Segundo Sommerville apud Piekarski e Quináia (2000), o objetivo da engenharia reversa é derivar o projeto ou especificação de um sistema, partindo-se de seu código fonte. A

reengenharia tem como objetivo produzir um sistema novo com maior facilidade de manutenção e a engenharia reversa é usada como parte do processo de reengeharia, pois fornece o entendimento do sistema a ser reconstruído (PIEKARSKI ; QUINÁIA, 2000).

A necessidade de utilizar a reengenharia está relacionada com a possibilidade de melhorar o entendimento de um software, prepará-lo ou melhorá-lo, visando aumentar sua manutenibilidade, seu reuso e sua extensão (BRAGA, 206). Também existem fatores que são relacionados com a complexidade das atividades empresariais, tais como a busca por produtividade, flexibilidade frente as constantes mudanças, a concentração no ramo de negócios, relacionamento com os clientes, meio ambiente e governos, além da parceirização e a gestão e remuneração dos recursos humanos por resultados. Todos esses fatores influenciam diretamente na reengenharia dos softwares existentes em empresas (HAMMER ; CHAMPY apud PIEKARSKI ; QUINÁIA, 2000). Além disso, todos os sistemas possuem um tempo de vida limitado, sendo que cada alteração efetuada pode degenerar sua estrutura, fazendo com que a evolução de um software nos processos de manutenção se torne cada vez mais difíceis (JACOBSON ; LINDSTROM apud PIEKARSKI e QUINÁIA, 2000).

Um cenário comum onde a reengenharia é aplicada pode ser vista em sistemas que servem a uma atividade de negócios durante anos (sistemas legados), e ao longo deste tempo o sistema tem sido adaptado e aprimorado muitas vezes, porém deixando de lado as boas práticas de Engenharia de Software. Dessa forma a aplicação continua em uso, porém instável. Assim, a cada mudança efeitos colaterais surgem, mas ainda assim a aplicação tem que continuar evoluindo. São nesses tipos de software, cuja facilidade de manutenção tem sido posta em segundo plano, que a reengenharia é aplicada (LEITÃO, 2001).

Necessidades de mudança são constantes em todo o trabalho de produção de software. Os mecanismos têm que ser desenvolvidos para avaliar, controlar e executar modificações de uma forma que o impacto no sistema seja o menor possível. Resumidamente a reengenharia resume-se em sete passos: documentar os softwares atuais; melhorar a leitura do código; redesenhar as bases de dados; alterar a plataforma de hardware; converter linguagens; adicionar novas funcionalidades e/ou capacidade; facilitar os processos de manutenção e fazer

evoluir os softwares em um ambiente CASE (tecnologia que inclui ferramentas de análise, projeto, desenvolvimento e análise de performance) (LEITÃO, 2001).

Aplicando essas técnicas a reengenharia trás alguns benefícios, como por exemplo, redução de custo, melhoramento da performance dos processos e aproveitamento dos chamados *breakthroughs,* que são entendidos como sendo algo que permite alcançar uma vantagem competitiva significativa e percebida pelo mercado como um elemento diferenciador (LEITÃO, 2001).

Apesar de existirem formas para realizar a reengenharia de software, estas dependem de muitos fatores, como por exemplo, forma de trabalho da empresa, a origem do software em questão e o desenvolvimento de tecnologias para apoiar essa atividade. A reengenharia de software não trata apenas de modernizar o software, mas também de adaptá-lo de acordo com as novas necessidades do processo em que está inserido (PIEKARSKI ; QUINÁIA, 2000).

3.6 CONCLUSÃO

Os estudos feitos neste capítulo mostram que através das metodologias apresentadas, é possível estruturar formas adequadas para se evoluir um software. Através das Leis de Lehman, consegue-se obter uma visão do estado que um software se encontra, através das características encontradas nele pela relação com as leis de evolução.

Definir a forma de realizar a manutenção pode levar a um processo mais estruturado de evolução, de forma que sejam selecionadas as atividades corretas que devem ser desempenhadas. A engenharia reversa e a reengenharia são atividades que dependendo da situação podem ser utilizadas para melhorar a compreensão do sistema que necessita ser evoluído.

Através desses entre outros métodos a tentativa de tornar mais longa a serventia de um software pode ser feita de forma menos árdua, e mais estruturada e organizada de se operar.

4 APLICAÇÃO DE MÉTODOS E TECNOLOGIA DE EVOLUÇÃO

4.1 INTRODUÇÃO

Trabalhar com manutenção de software com o objetivo de evoluir sistemas nem sempre é uma tarefa fácil. Neste capítulo é apresentada a gerência de configuração de sistemas que tem por objetivo controlar a evolução de sistemas de software. Além disso, também são identificadas ferramentas de controle de versão e modificação que fornecem suporte para realizar a manutenção de sistemas de forma mais controlada, com mais qualidade e rapidez.

A necessidade de evoluir sistemas é cada vez maior em ambientes que sofrem algum tipo de alteração nas regras de negócio e nos meios de trabalho. Para que essas modificações sejam feitas de forma organizada e sejam controladas de uma melhor forma, é necessário adotar técnicas e ferramentas que auxiliam na evolução de software. Isso pode ser alcançado através da adoção de Gerência de Configuração de software e ferramentas de controle de modificação e versão.

Este capítulo apresenta conceitos de gerência de configuração de software e apresenta ferramentas que junto a esta técnica auxiliam na execução dos processos de evolução de software.

4.2 GERÊNCIA DE CONFIGURAÇÃO DE SOFTWARE

Durante a vida de um software, o processo de manutenção de software é uma atividade inevitável e que chega a consumir 75% do custo total do ciclo de vida (BLANCHARD apud DANTAS et al., 2003). Dessa forma torna-se necessário a utilização de mecanismos que possibilite o controle da mudança, pois ambientes de desenvolvimento que não controlam as mudanças são levados rapidamente ao caos (PRESSMAN, 2006). A utilização da Gerência de Configuração de Software pode ser útil para melhor organizar os processos de mudança.

A Gerência de Configuração de Software (GCS) é classificada como sendo uma disciplina para o controle da evolução de sistemas de software (DART apud MURTA, 2007).

Já Cunha et al. (2004) classifica o Gerenciamento de Configuração de Software como um processo que possui recursos para identificar, controlar a evolução e realizar a auditagem dos artefatos de software construídos durante o desenvolvimento do projeto de software. E ainda, Burrows apud Dantas et al. (2003), diz que GCS é uma abordagem disciplinada para gerenciar o processo de evolução do desenvolvimento e manutenção do software.

Dessa forma, a GCS não se propõe a definir quando e como devem ser executadas as modificações nos artefatos de software, que já é função do próprio desenvolvimento de software (MURTA, 2004).

O objetivo da GCS é estabelecer e manter a integridade dos produtos do projeto de software ao longo do seu ciclo de vida (CUNHA et al., 2004). A GCS pode ser tratada sob diferentes perspectivas, na gerencial ela é dividida em quatro funções que são: identificação da configuração, controle da configuração, acompanhamento da configuração e auditoria da configuração (MURTA, 2004).

A função de identificação dá início ao controle sobre os itens de configuração (IC) selecionados. Um IC pode ser qualquer artefato que demanda GCS. É na função de identificação que as tarefas referentes a seleção e documentação dos ICs são executadas, sendo que podem estar em diferentes níveis de abstração (análise, projeto, codificação, teste, etc) (MURTA, 2004).

A função de controle da configuração é responsável por autorizar, implementar e verificar as modificações sobre os ICs. O controle ainda é composto por tarefas, que são formados por requisição de modificação, onde é criada utilizando um sistema de controle de modificação, em seguida é feita a classificação da requisição de modificação de acordo com o grau de relevância, depois é feito o processo de análise de impacto que detecta o custo e o cronograma relacionados a modificação. A partir daí, o comitê responsável por avaliar a requisição de modificação, o *Change Control Board,* avalia qual o destino da requisição de modificação, caso seja aprovada é iniciada a implementação, utilizando um sistema de controle de versão, caso não seja aprovado o motivo deve ser explicitado ao solicitante. Ao final deve ser feita a verificação sobre o que foi implementado para assegurar a inconsistência da nova *baseline* (MURTA, 2004).

A função de acompanhamento armazena as informações produzidas nas demais funções e relata essas informações aos desenvolvedores interessados e autorizados. E ainda há a função de auditoria que gera *releases,* versão de *baseline* a ser entregue ao cliente, onde é verificado a corretude da *release*, examinando planos, dados e resultados de teste, além de verificar através da completude da *release*, se os artefatos definidos no início do projeto estão sendo entregues. A ilustração 4 mostra o processo descrito anteriormente (MURTA, 2004).

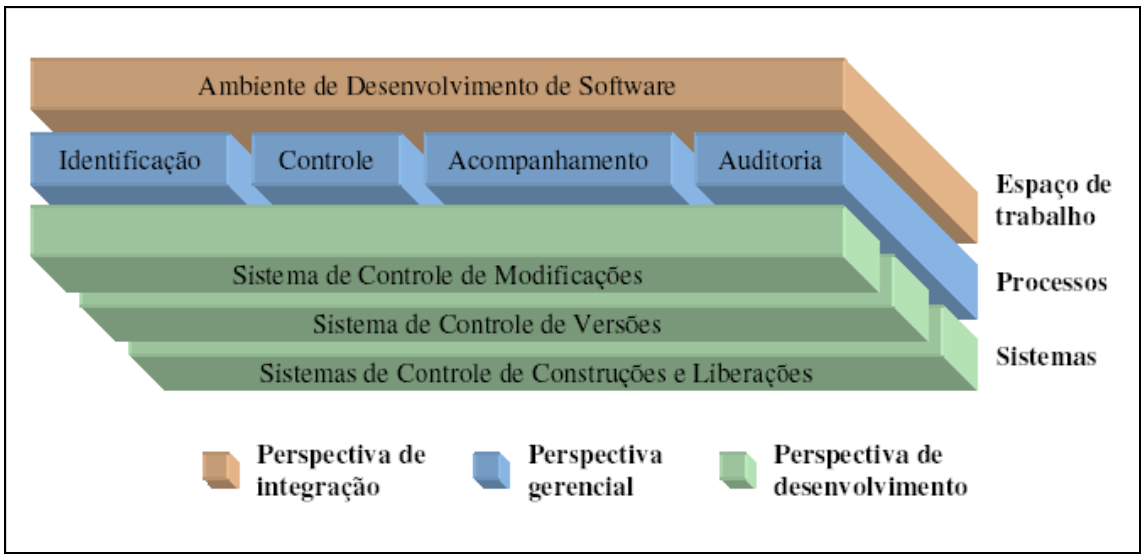

ILUSTRAÇÃO 4 - Perspectivas de GCS
Fonte: MURTA, 2004.

Os sistemas de GCS são muito úteis para controlar de forma mais estruturada e planejada as manutenções que um software pode vir a sofrer, além de que, está técnica ainda utiliza o sistema de controle de versão e de modificação podendo auxiliar no controle das modificações e também sobre relações de históricos feitos sobre alterações efetuadas (DANTAS et al., 2003).

4.3 FERRAMENTAS PARA AUXILIAR O CONTROLE DA EVOLUÇÃO DE SISTEMAS

Com a grande competitividade na área de desenvolvimento de software, torna-se necessário garantir a qualidade dos produtos de software (ROCHA apud

FORTES, 2005 A). É inevitável que modificações surjam durante a construção de um software. Essas modificações devem ser analisadas antes de serem executadas, registradas antes de serem implementadas, relatadas àqueles que tem necessidade de saber delas e controladas no sentido de melhorar a qualidade e reduzir os erros para que não se tornem um problema (PRESSMAN, 2006). Para auxiliar nessa tarefa existem ferramentas que ajudam no controle do desenvolvimento e da evolução o software. No entanto, existem ferramentas que fazem o controle de versões e ferramentas que fazem o controle de modificações. Sendo que estas podem ser implementadas para trabalharem juntas.

Dentre as ferramentas que fazem o controle de versão estão, Aegis, GNU Arch, CVS e Subversion, que são ferramentas livres. Além das livres também existem as comerciais que são, BitKeeper (BitMOver), ClearCase (IBM Rational), Perforce, PVCS (Serena), Star Team (Borland), Synergy/CM (Telelogic) e Visual Studio Team Foundation (Microsoft) entre outras. Já as ferramentas utilizadas para o controle de modificações são, Mantis, Trac e Bugzilla, que são ferramentas livres, e as comerciais são, ClearQuest (IBM Rational), JIRA, Star Team (Borland), Synergy/Change (Telelogic), TeamTrack (Serena) e Visual Studio Team Foundation (Microsoft) entre outras (MURTA, 2007).

Realizar o controle de versão é importante, já que software é algo caro a ser produzido, toma muito tempo e exige organização e cooperação de várias pessoas. Dessa forma é importante armazenar tudo o que for feito. Uma ferramenta de controle de versão tem como função, armazenar e ordenar diversas versões de um documento ou projeto. Através dela é possível manter um histórico das mudanças efetuadas, analisar as diferenças entre versões além de permitir o trabalho cooperativo (FORTES, 2005 B).

As vantagens em se utilizar uma ferramenta de controle de versões são inúmeras. Uma dessas vantagens é o armazenamento centralizado onde é possível ser feito um controle de acesso às informações, cópia de segurança e distribuição da mesma. Uma outra vantagem é a possibilidade de trabalhar com histórico universal de revisões, com isso toda alteração realizada é registrada de forma que nada que foi alterado seja perdido e as versões anteriores ainda podem ser acessadas rapidamente. Também é possível fazer uma análise entre diferentes versões verificando suas diferenças e identificando quem realizou as alterações (RAVAZI, 2005).

Uma outra questão importante é o trabalho cooperativo, onde vários desenvolvedores trabalham sobre o mesmo conjunto de arquivos, de forma que suas alterações feitas são mescladas, permitindo assim maior controle sobre o que está sendo modificado. Pelo controle de ramificação, é possível fazer alterações em uma cópia do documento sem alterar o original de forma que o documento original continue recebendo outras alterações, sendo que as alterações das ramificações podem ser mescladas com a do documento original. No controle de versões é possível armazenar tudo o que é produzido manualmente, como por exemplo, código-fonte, scripts, documentação escrita, figuras, imagens, ícones e makefiles. Porém existem arquivos que não devem ser armazenados no controle de versões, como arquivos gerados por processos automáticos, como por exemplo, código-objeto, programas compilados e documentação produzida automaticamente. Também não devem ser armazenados arquivos com configurações locais como nome e senha de acesso a banco de dados, e arquivos criados acidentalmente como os arquivos temporários (RAVAZI, 2005).

Dentre os sistemas de controle de versão citados anteriormente, o mais utilizado é o CVS que já esta sendo substituído pelo Subversion, que além de realizar as mesmas funções do CVS, ainda é mais implementado, e executa tarefas que são limitadas ao CVS (RAVAZI, 2005).

No controle de modificação para realizar o armazenamento e a notificação do que está sendo modificado, existem tarefas a serem executadas, como solicitação de modificação, classificação da modificação, análise da modificação, avaliação da modificação, implementação da modificação, verificação da modificação e geração de *baseline*. Para realizar devidas alterações, critérios de classificação de modificações devem ser explicitados no planejamento. Através dessas classificações é possível priorizar quais modificações são mais importantes para que sejam realizadas primeiro. Essas classificações podem ser de nível crítico, fatais, não fatais e cosméticas. Também é necessário realizar análises sobre os impactos da modificação. Essas análises envolvem custo, cronograma e funcionalidades. Caso a análise não aprove a modificação, o que é raro de ocorrer, pode ocorrer rejeição antes da avaliação para poupar custos. Isso pode ser feito para poupar custos no processo. Após a análise é feita uma avaliação que utiliza a solicitação de modificação e o laudo da análise para tomar a decisão de aceitar, rejeitar ou adiar a modificação. Essa decisão é tomada pelo Comitê de Controle da

Configuração (CCC), compostos por líderes do projeto e pela gerência de configuração. Caso seja aprovada, a modificação deve ser seguida por testes de unidade. Durante a verificação, devem ser aplicados testes de sistema, e finalmente, após a geração da nova baseline, que é a configuração revisada e aprovada que serve como base para uma próxima etapa de desenvolvimento, o CCC decide se ela é considerada uma nova liberação (MURTA, 2007).

Percebe-se que tanto o controle de modificação quanto o de versão possuem tarefas distintas, com objetivos semelhantes. Dessa forma a união das duas tecnologias faz com que o processo de desenvolvimento e manutenção seja realizado de forma mais estruturada, com mais qualidade e organização estabelecendo mais eficiência na execução das atividades evolutivas de um software.

4.4 CONCLUSÃO

Pelo que foi estudado neste capítulo, pode-se concluir que um processo de evolução de software pode ser muito complexo e as manutenções seguidas neste processo demandam custo e tempo. Porém, a utilização de técnicas como a de Gerência de Configuração, pode auxiliar no controle das mudanças, de forma a simplificar o processo de modificação de um software, tornando-o através de planejamentos, mais organizado e estruturado.

Conclui-se também que, em um processo de modificação de software, é possível utilizar junto a essas técnicas, ferramentas que tornam o processo de modificação mais controlado, com maior produção em menor tempo. Tais ferramentas auxiliam no controle de versão e modificação de um software, onde estas podem ser integradas a fim de aumentar a eficiência de trabalho e tornar maior eficácia nos resultados.

5 CONSIDERAÇÕES FINAIS E PERSPECTIVAS FUTURAS

A necessidade de automatizar processos em empresas, organizações e em instituições vem tornando o desenvolvimento de software mais constante, com isso, exige-se cada vez mais qualidade do que está sendo construído, independente da utilização.

Na maioria das vezes os softwares são construídos sem a preocupação das novas demandas que vão realizar futuramente. Equipes de desenvolvimento de software, na maioria das vezes não adotam medidas corretas no desenvolvimento, seja por falta de conhecimento ou mesmo por problemas encontrados nos processos de manutenção. Para que isso não ocorra medidas de evolução de software devem ser adotadas, já que mais cedo ou mais tarde, os softwares têm de sofrer manutenções para que possam evoluir e atender a novas necessidades do ambiente em que atuam.

Apesar de ser um tema novo e que exige mais pesquisas, a evolução de software deve ser levada em consideração sobre os processos de desenvolvimento na tentativa de “educar” a forma de construir softwares. Identificar o momento em que sistemas precisam evoluir, recuperar softwares legados e adiar o fim de determinados sistemas, são medidas consideradas pela evolução de software.

Este trabalho contribui no que diz respeito à apresentação dos conceitos, das etapas, dos métodos, de algumas ferramentas que podem ser utilizadas para apoiar na evolução softwares.

Quanto à relevância, este trabalho mostra que a evolução de software se torna necessária para buscar o rejuvenescimento de softwares, através da observação do comportamento dos softwares ao longo do tempo, encontrar alternativas para que a utilidade do software seja prolongada.

A sugestão para trabalhos futuros é elaborar estudos considerando aspectos de rejuvenescimento dos sistemas, técnicas como a reutilização de software, trabalhar as leis de Lehman sobre a evolução de softwares livres, para que dessa forma torne mais eficiente e produtivo o desenvolvimento de softwares.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, J. **Mudança de software**. Universidade Nova de Lisboa, 2005. Disponível em: <http://www-ctp.di.fct.unl.pt/~ja/mei-es/mud.ppt.> Acesso em: 26 abr.2007.

ARAÚJO, M. A. P., TRAVASSOS, G. H. **Em busca de um *Framework* para estudos experimentais em evolução de software**. Rio de janeiro: COPPE/UFRJ, 2004.

ARAÚJO, M. A. P., TRAVASSOS, G. H. **Em busca de um *Framework* para estudos experimentais em evolução de software**. In: WORKSHOP DE MANUTENÇÃO DE SOFTWARE MODERNA, 3, 2006, Vila Velha, **Anais**... Vila Velha: WMSWM, 2006.

BARBOSA, N. M., PERKUSICH, A. **Estudo experimental comparativo de modelos de componentes para o desenvolvimento de software com suporte à evolução dinâmica e não antecipada**. Campina Grande: UFCG, 2006.

BONACIN, R. **Engenharia de software**. UNICAMP: 2006. Disponível em: <http://www.ceset.unicamp.br/webdidat/matdidat.php?cod=ST062&nome=Rodrigo+Bonacin>. Acesso em: 19 set. 2007

BRAGA, R. T. V. **Engenharia reversa e reengenharia**. Paraná: UFPR, 2006. Disponível em: <http://www.inf.ufpr.br/silvia/ES/reengenharia/reengenharia.pdf>. Acesso em: 26 out. 2007.

CANHOTA, A. J. S. da. **Engenharia reversa.** Rio de Janeiro: UFF, 2005. Disponível em: <http://www.ic.uff.br/~otton/graduacao/informaticaI/apresentacoes/eng_reversa_slides.pdf>. Acesso em: 26 ago. 2007.

CHRISTOPH, R. de H. **Engenharia de software para software livre**. Pontifícia Universidade Católica do Rio de Janeiro: 2004. Cap. 3, p.36-50. Disponível em: <http://www.2.dbd.puc-rio.br/pergamum/tesesabertas>. Acesso em: 26 mar. 2007.

CUNHA, J. R. D et al. **Uma abordagem para o processo de gerenciamento de configuração de software.** São Carlos: UFSCAR, 2004. Disponível em: <http://www.inf.ufsc.br/resi/edicao04/artigo05.pdf>. Acesso em: 01 nov. 2007.

DANTAS, C. R et al. **Um estudo sobre gerência de configuração de software aplicada ao desenvolvimento baseado em componentes**. Rio de Janeiro: COPPE/UFRJ, 2003. Disponível em: <http://reuse.cos.ufrj.br/prometeus/publicacoes/wdbc03-gcs.pdf>. Acesso em: 01 nov. 2007

FELTRIM, V. D. **Apoio à documentação de engenharia reversa de software por meio de hipertextos.** São Carlos: USP, 1999. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-01022001-172954/>. Acesso em: 26 out. 2007

FORTES, R. P. de M. **Experiências com uso de Bugzzila – lições sobre a adoção de processo de desenvolvimento e evolução de software**. 2005 A. Disponível

em: <http://safe.icmc.usp.br/safe/technical-reports/9896.pdf/view>. Acesso em: 02 nov. 2007

FORTES, R. P. de M. **Integrando controle de versões e de alterações**. 2005 B. Disponível em: <http://safe.icmc.usp.br/safe/technical-reports/9866.pdf/view>. Acesso em: 02 nov. 2007.

LEITÃO, M. A. A. **Engenharia reversa na engenharia de software.** Rio de Janeiro:UFRJ, 2001.Disponível em: <http://www.dcc.ufrj.br/~schneide/es/2001/1/g18/Engenharia%20Reversa.htm>. Aceso em: 29 out. 2007

LEITE,J. C. S. do P. **Evolução de software**. In: SBES, 20., 2006, Florianópolis, **Anais**... Florianópolis, SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 2006. p. 1-3.

LEITE,J. C. S. do P. **Gerenciando a qualidade de software com base em requisitos**. São Paulo: 2001. Cap. 17. Disponível em: <http://www.inf.puc-rio.br/~julio>. Acesso em: 17 mai. 2007

LOCATELLI, M. H.; KOMOSINSKI, L. J. **Engenharia de software para o desenvolvimento de WEBAPPS e as metodologias OOHDM e WEBML**.Florianópolis: Universidade Federal de Santa Catarina: 2003. Cap. 2, p.16-19. Disponível em: <http://www.inf.ufsc.br/~leandro/ensino/esp/monografia MarcioHenriqueLocatelli.pdf>. Acesso em: 9 set. 2007.

MARTINS, E. **Evolução de software**. Campinas: UNICAMP, 2001. Disponível em: <http://www.ic.unicamp.br/~eliane/Cursos/Transparencias/Manutencao/ evolucao1.pdf>. Acesso em: 20 mar. 2007.

MURTA, L. G. P. **Manutenção de software.** XII SEMANA DE INFORMÁTICA, 2007, Juiz de Fora. **Anais**... Juiz de Fora, CESJF, 2007. 51 p.

MURTA, L. G. P. **ODYSSEY-SCM: uma abordagem de gerência de configuração de software para o desenvolvimento baseado em componentes**. Rio de Janeiro: UFRJ, 2004. Disponível em: <http://reuse.cos.ufrj.br/prometeus/ publicacoes/odyssey-scm.pdf>. Acesso em: 01 nov. 2007

PIEKARSKI, A. E. T., QUIANÁIA, M. A. **Reengenharia de software: o que, por quê e como.** Guarapuava: UNICENTRO, 2000. Disponível em: <http://www.unicentro.br/ editora/revistas/recen/v1n2/Reengenharia.pdf>. Acesso em: 30 out. 2007.

PRESSMAN, R. S. **Engenharia de software**. 3. ed. São Paulo: Pearson Education, 2006. 1028 p.

SOUZA, T. B. A. de. **Um modelo para a avaliação da manutenibilidade de código-fonte orientado a objeto.** Pernambuco: UFPE, 2005. Disponível em: <http://www.cin.ufpe.br/~tg/2005-1/tbas.pdf>. Acesso em: 30 set. 2007.

TORQUATO, J. R. C**. Rápida análise da evolução de software**. Rio de Janeiro: PUC, 2006. Disponível em: <http://ritomar.blogspot.com/>. Acesso em: 3 out. 2007.