RELATÓRIO TÉCNICO
ES-641/04

Em Busca de um *Framework* para Estudos Experimentais em Evolução de Software

Marco Antonio Pereira Araújo
Guilherme Horta Travassos

Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ
http://www.cos.ufrj.br

Abril/2004

1. Introdução

"Programas, assim como as pessoas, envelhecem. Não podemos deter o envelhecimento, mas podemos entender suas causas, tomar atitudes para limitar seus efeitos, temporariamente reverter alguns dos danos causados, e preparar-nos para o dia em que o software não será mais viável" (PARNAS, 1994).

Sistemas de software estão crescendo rapidamente em termos de funcionalidade, complexidade, tamanho e estrutura (LEHMAN, 1998). Portanto, torna-se necessário antecipar os caminhos em que o software sofre mudanças, assim pode-se modificá-lo mais facilmente para acomodar estas necessidades. Entretanto, antecipar-se às mudanças não é uma tarefa fácil, uma vez que existem muitas razões pelas quais os sistemas mudam (PFLEEGER, 1998).

Enquanto a manutenção se refere às atividades que acontecem em qualquer época após a implementação de um novo projeto de desenvolvimento de software, a evolução de software é definida pelo exame do comportamento dinâmico das características dos sistemas, como eles mudam ao longo do tempo.

As Leis de Evolução de Software descrevem como um sistema se comporta ao longo de suas sucessivas versões (LEHMAN, 1980). Trabalhos encontrados na literatura referenciam estudos experimentais em evolução de software considerando apenas o código fonte de sistemas legados (KEMERER & SLAUGHTER, 1999) (SCACHI, 2003). Além disso, LEHMAN & RAMIL (2002) têm apontado a necessidade de estudos de evolução em sistemas orientados a objetos e em outros níveis de abstração no processo de desenvolvimento de software (LEHMAN & RAMIL, 2003). Em virtude destas características, torna-se importante o estudo das causas de decaimento ao longo de processos de desenvolvimento OO, proporcionando-nos um melhor entendimento de como este tipo de software evolui.

Assim, este trabalho irá apresentar o tema Evolução de Software no contexto da Engenharia de Software Experimental, apresentando os conceitos e características envolvidas na área, bem como uma revisão bibliográfica sobre o assunto, com ênfase em estudos experimentais. KEMERER & SLAUGHTER (1999) afirmam que apenas 2% dos estudos experimentais focam em manutenção, a despeito de que publicações reportam que ao menos 50% do esforço de software é dedicado a esta atividade. Dentre as publicações encontradas na literatura, a maioria refere-se a estudos de observação, tratando normalmente de evolução em código fonte de sistemas legados, alguns ainda referentes a sistemas em *batch* e, normalmente, na linguagem COBOL.

Este trabalho ainda apresenta as Leis de Evolução de Software como a base para uma teoria de evolução de software, originalmente proposta por LEHMAN (1980). De acordo com SCACHI (2003), esta teoria representa uma das maiores contribuições intelectual e apresenta desafios para a comunidade de pesquisa em evolução de software.

Baseado nestas Leis de Evolução de Software e, tendo em vista não ter sido encontrado na literatura técnica material relativo ao estudo da evolução de software para sistemas utilizando o paradigma da orientação a objetos, seja para codificação ou tampouco relativo a outras etapas do processo de desenvolvimento, entendemos como sendo uma maior contribuição deste trabalho a elaboração da hipótese de que as leis de

evolução de software podem também ser suportadas pelas diferentes etapas de um processo de desenvolvimento de software utilizando o paradigma da orientação a objetos, invés de apenas à fase de codificação de sistemas legados.

Neste sentido, o objetivo deste trabalho é apresentar a pesquisa que vem sendo realizada para a elaboração de um framework para estudos de evolução de software no contexto de processos de desenvolvimento utilizando o paradigma da orientação a objetos, baseado nas Leis de Evolução de Software propostas em (LEHMAN, 1980). A finalidade é a elaboração de uma estrutura conceitual que apóie a definição de um conjunto de estudos experimentais para o estudo do decaimento de software baseado nas Leis de Evolução em diferentes níveis de abstração de processos de desenvolvimento de software orientado a objetos.

Este trabalho é dividido em mais seis seções além desta introdução. A seção 2 apresenta a conceituação referente à Evolução de Software. A seção 3 trata das Leis de Evolução de Software. Na seção 4 são apresentados os principais estudos experimentais sobre Evolução de Software. A seção 5 apresenta uma discussão inicial sobre a aplicação das Leis de Evolução de Software em processos de desenvolvimento de software orientado a objetos, apresentando um framework para estudos de evolução neste contexto. A seção 6 apresenta as hipóteses a serem verificadas experimentalmente utilizando o método GQM (*Goal / Question / Metric*) (BASILI & WEISS, 1984) (SOLIGEN & BERGHOUT, 1999) e a seção 7 apresenta as considerações finais e perspectivas de trabalhos futuros.

2. Evolução de Software

Sistemas de software estão crescendo rapidamente em termos de funcionalidade, complexidade, tamanho e estrutura (LEHMAN, 1998). Portanto, torna-se necessário antecipar os caminhos em que o software sofre mudanças, assim pode-se modificá-lo mais facilmente para acomodar estas necessidades. Entretanto, antecipar-se às mudanças não é uma tarefa fácil, uma vez que existem muitas razões pelas quais os sistemas mudam (PFLEEGER, 1998).

KEMERER & SLAUGHTER (1999) fazem uma distinção entre manutenção e evolução de software:

- Manutenção: a correção de erros e a implementação de modificações necessárias para permitir a um sistema existente a executar novas tarefas, e para executar antigas sob novas condições.
- Evolução: o comportamento dinâmico dos sistemas, como eles são mantidos e expandidos ao longo de seu ciclo de vida.

Enquanto a manutenção se refere às atividades que acontecem a qualquer época após a implementação de um novo projeto de software, a evolução de software é definida pelo exame do comportamento dinâmico das características dos sistemas, como elas mudam ao longo do tempo.

A literatura apresenta um conjunto de termos comumente associados com evolução de software. Deterioração, decaimento e envelhecimento referem-se aos problemas causados pela evolução do software. O termo rejuvenescimento é utilizado no sentido de reverter estes problemas.

EICK et al. (1999) apresentam um estudo indicando que código decai quando é mais difícil modificá-lo do que deveria ser. Esta afirmação baseia-se no custo da mudança, que é efetivamente o custo associado aos desenvolvedores; no intervalo para completar a mudança (o tempo requerido); e a qualidade do software modificado.

No mesmo estudo, EICK et al. (1999) afirmam que a mudança de código é a causa para o decaimento de software e apresenta uma série de causas para este decaimento:

1. Arquitetura inapropriada que não suporta as mudanças ou abstrações requeridas para o sistema;
2. Violações dos princípios originais do projeto, que pode forçar mudanças não previstas ou não obedecer as suposições originais;
3. Requisitos imprecisos, que podem impedir programadores de desenvolver códigos corretos, causando excessivo número de mudanças;
4. Pressões de tempo, que podem levar desenvolvedores a produzir código de baixa qualidade ou fazer mudanças sem o entendimento do impacto no sistema;
5. Ferramentas inadequadas de programação, isto é, indisponibilidade de ferramentas CASE;
6. Ambiente organizacional manifestado, por exemplo, em baixa estima, rotatividade excessiva, inadequada comunicação entre os desenvolvedores, tudo que possa produzir frustração e prejudicar o trabalho;
7. Variabilidade do programador, ou seja, programadores que não entendem ou fazem delicadas mudanças em código complexo escrito pelos colegas mais experientes;

8. Processo de mudança inadequado, como a falta de controle de versões ou inabilidade para conduzir mudanças em paralelo.

Como decorrência das causas de decaimento, EICK et al. (1999) definem sintomas como manifestações mensuráveis de decaimento, apresentando uma série de sintomas de decaimento de software:

1. Complexidade excessiva;
2. Histórico de mudanças freqüentes;
3. Histórico de falhas;
4. Mudanças amplamente dispersas;
5. Impropriedades no código quando desenvolvedores fazem mudanças que deveriam ter sido feitas de forma mais elegante ou eficiente;
6. Interfaces numerosas.

EICK et al. (1999) ainda definem um conjunto de fatores de risco para decaimento de software, considerando que fatores de risco incrementam a probabilidade de decaimento de software ou multiplicam seu efeito:

1. O tamanho de um módulo, em NCSL (número de linhas de código fonte não comentadas);
2. A idade de um código (conceito intuitivo);
3. Complexidade inerente;
4. Rotatividade ou reorganização (degradação da base de conhecimento), podendo aumentar a probabilidade de desenvolvedores inexperientes fazendo mudanças em código;
5. Código portado ou reutilizado, originalmente desenvolvido em uma outra linguagem, para um sistema diferente ou para outra plataforma de hardware;
6. Carga de requisitos, significando que o código tem funcionalidade extensiva e está sujeito a muitas restrições (dificuldades de entendimento e de implementação);
7. Desenvolvedores inexperientes, pela falta de conhecimento em desenvolvimento, falta de entendimento da arquitetura do sistema, e potencial para baixa habilidade em desenvolvimento.

EICK et al. (1999) definem Índices de Decaimento de Software (CDI – *Code Decay Indices*) como interpretações de sintomas e fatores de risco quantificados, ou prognósticos, que são indicativos dos resultados (custo, intervalo e qualidade), computáveis diretamente através de métodos estatísticos:

- Histórico de Mudanças Freqüentes: quantidade de mudanças por período de tempo;
- Quantidade de Mudanças: número de arquivos modificados em função de uma mudança (indica quebra de encapsulamento e modularidade);
- Tamanho: número de linhas não comentadas de um módulo;
- Idade: idade média das linhas constituintes de um módulo;
- Potencial de Falhas: quantifica sintomas de fatores de risco (em função da quantidade de linhas incluídas e excluídas por unidade de tempo);
- Esforço: prediz o esforço (homem/hora) requerido para implementar uma mudança (em função das quantidades de linhas incluídas e excluídas por unidade de tempo, número de arquivos modificados, número de desenvolvedores envolvidos e tempo requerido para implementação).

Os resultados deste estudo demonstram (EICK et al., 1999):

- Incremento, ao longo do tempo, do número de arquivos em virtude da mudança de código;
- Declínio na modularidade de um subsistema de código, medido pelas mudanças em múltiplos módulos;
- Contribuições de vários fatores (particularmente, freqüência de mudanças) para a taxa de falhas em módulos de código;
- Quantidade e tamanho das mudanças são fatores importantes para predizer o esforço de implementar uma mudança.

PARNAS (1994) aponta sintomas de envelhecimento de software como dificuldades de atualização de produtos para o mercado e conseqüente perda de clientes, perda de desempenho como resultado de uma estrutura gradualmente deteriorada e apresentação de erros introduzidos quando mudanças são feitas.

PFLEEGER (2004) trata de rejuvenescimento de software como o desafio da manutenção no sentido de tentar aumentar a qualidade de um sistema existente. Ou seja, minimizar os problemas causados pelo decaimento de software. Existem diversos aspectos do rejuvenescimento de software a serem considerados, como redocumentação, reestruturação, engenharia reversa e reengenharia.

3. As Leis de Evolução de Software

O trabalho de LEHMAN (1980) é um dos precursores da área de evolução de software e descreve um sistema em termos dos caminhos em que se relaciona com o ambiente em que opera, sendo sucedido por vários trabalhos que relatam a evolução de sistemas, iniciando no início da década de 70. Os estudos originais foram baseados na evolução do IBM OS/360, seguido de vários outros estudos, inclusive com avaliação na indústria (LEHMAN et al., 1997, 1998, 2000, 2001).

As Leis de Evolução de Software e seu desenvolvimento como a base para uma teoria de evolução de software representam a maior contribuição intelectual e desafios para a comunidade de pesquisa em evolução de software (SCACCHI, 2003).

Lehman propôs a classificação dos sistemas em três tipos, evidenciando que a natureza de um sistema determina a forma de evolução do mesmo:

- S-Systems (*specification*): o problema é bem definido e a solução é bem conhecida. Está diretamente relacionado com o mundo real e, se este muda, o resultado é um problema completamente novo que deve ser especificado assim, são improváveis de mudar. Sistemas para operações em matrizes são exemplos de sistemas deste tipo. Este tipo de sistema está esquematizado na figura 3.1;

- P-Systems (*problem*): baseia-se uma abstração prática do problema, ao invés de uma especificação completamente definida, sendo mais dinâmico que um S-System. A solução produz informação que é comparada com o problema e depende em parte do analista que gerou os requisitos, sendo sujeitos à mudanças incrementais. Sistemas para jogo de xadrez são exemplos de sistemas deste tipo. Este tipo de sistema está esquematizado na figura 3.2;

- E-Systems (*embedded*): incorpora mudanças naturais do mundo real, uma vez que está embutido no mundo real e muda com ele. A solução é baseada no modelo do processo abstrato envolvido. Como o sistema é uma parte do mundo modelado, são provavelmente submetidos à mudanças quase constantes. Sistemas para a área financeira são exemplos de sistemas deste tipo. Este tipo de sistema está esquematizado na figura 3.3.

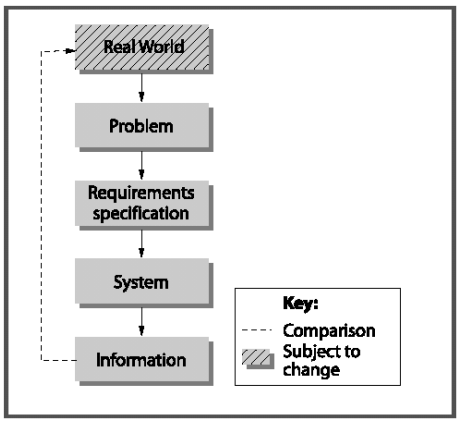

Figura 3.1 – Esquematização de um S-System

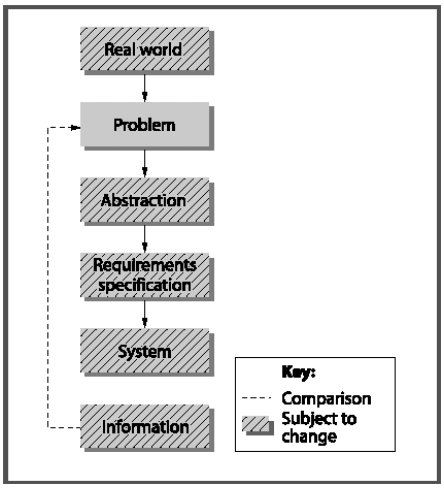

Figura 3.2 – Esquematização de um P-System

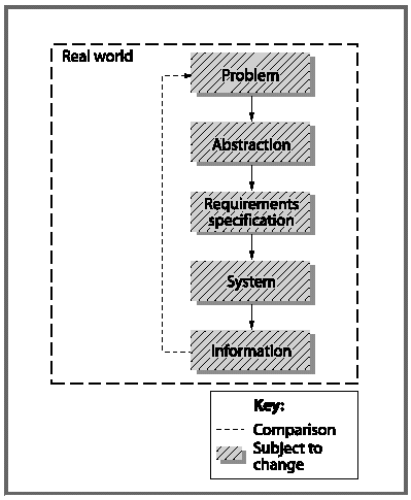

Figura 3.3 – Esquematização de um E-System

A contribuição principal do trabalho de Lehman está na elaboração das Leis de Evolução de Software e, segundo o próprio autor, “as novas análises suportam, ou melhor, não contradizem, as leis de evolução de software, sugerindo que a abordagem da década de 70 para analisar as medidas de evolução de software é ainda relevante atualmente” (LEHMAN, 1998).

As Leis de Evolução de software, enunciadas a seguir, focam, primeiramente, no entendimento de como sistemas de software mudam ao longo do tempo (LEHMAN et al, 1998).

POSTULADAS EM 1974:

I - Mudança Contínua

- 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;

II - Incremento da Complexidade

- 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.

III - Auto-Regulação

- 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 medidas e previstas.

POSTULADAS EM 1978:

IV - Conservação da Estabilidade Organizacional

- 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.

V - Conservação da Familiaridade

- 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.

VI - Crescimento Contínuo

- O conteúdo funcional de um sistema deve ser continuamente incrementado para manter a satisfação do usuário ao longo do ciclo de vida.

POSTULADA EM 1994:

VII - Declínio da Qualidade

- A qualidade de um sistema entrará em declínio a menos que seja rigorosamente mantida e adaptada às mudanças do ambiente operacional.

POSTULADA EM 1972 e MODIFICADA EM 1996:

VIII – Sistema de Realimentação

- 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 a alcançar significativas melhorias.

LEHMAN et al. (1998) ainda descrevem um conjunto de Métricas de Evolução em Manutenção de Software, no contexto do projeto FEAST/1. Este projeto surgiu do conjunto de hipóteses FEAST (*Feedback, Evolution And Software Technology*), formuladas em 1994. O projeto de pesquisa, iniciado em 1996, tem procurado por evidências tangíveis no processo de evolução de software na indústria e, neste sentido, tem considerado a seguinte lista de métricas e indicadores:

- Seqüência de número de versões;
- Tamanho do sistema (subsistemas, módulos, arquivos, etc.);
- Elementos tratados (um módulo com n mudanças independentes é contado n vezes);
- Elementos adicionados;
- Elementos modificados;
- Elementos apagados;
- Elementos em tratamento;
- Intervalo entre versões ou disponibilidade geral;
- Esforço aplicado (em unidades apropriadas);
- Erros detectados (por versão);
- Erros corrigidos (por versão).

Os conceitos de classificação de sistemas de Lehman, acompanhado de suas leis, provêem um vocabulário amplamente aceito para a discussão da natureza da mudança de software.

Através destas idéias, pode-se projetar sistemas para serem flexíveis, planejar manutenções, além de melhor entender e controlar o desenvolvimento de software, mais do que simplesmente reagir aos problemas que acontecem (PFLEEGER, 1998).

4. Estudos Experimentais em Evolução de Software

Vários estudos experimentais em evolução de software foram encontrados na literatura. Alguns apresentam os testes estatísticos ou a formulação matemática utilizados para suportar seus resultados. Entretanto, a maioria dos estudos encontrados tratam de observação, relatando características observadas durante vários anos de observação da evolução de alguns sistemas de software.

Alguns dos estudos apresentados foram desenvolvidos a partir da plataforma de software livre. Estes estudos demonstram que o software livre é uma área promissora para os estudos de evolução de software, uma vez que sistemas desenvolvidos com esta abordagem, além do código fonte disponível, também apresentam os registros de manutenções ocorridas como uma prática comum das comunidades que desenvolvem software nesta plataforma, oferecendo um rico conjunto de dados para estudos experimentais em evolução de software.

KEMERER & SLAUGHTER (1999) afirmam que apenas 2% dos estudos experimentais focam em manutenção, a despeito de que publicações reportam que ao menos 50% do esforço de software é dedicado a esta atividade.

A seguir estão descritas as principais características de alguns estudos experimentais relatados na literatura.

i) BELADY & LEHMAN (1976):

- trabalho experimental pioneiro na área de evolução de software;
- observações sobre crescimento de tamanho e complexidade de sistemas, que levaram a postular cinco leis de evolução de software (Mudança Contínua, Incremento da Complexidade, A Lei Fundamental da Evolução de Programas, Conservação da Estabilidade Organizacional e Conservação da Familiaridade), publicadas posteriormente em 1980;
- dados experimentais são apresentados relacionados aos 21 últimos releases de um sistema batch, suportando as leis postuladas.

ii) YUEN (1985, 1987, 1988):

- primeiros estudos sistemáticos das leis de Belady & Lehman;
- foram analisados por 19 meses dados de defeitos em um sistema de grande porte;
- sete variáveis dependentes foram descritas, mas apenas dois conjuntos de resultados, classe de prioridade (severidade) e tempo de resposta, são descritos;
- relata resultados relacionados à descoberta e correção de defeitos;
- relata ainda que o tempo gasto para corrigir um defeito não cresce com o tempo, ao contrário do que seria esperado, em virtude do crescimento da complexidade do sistema;
- constata de que as duas primeiras leis eram suportadas mas não conseguiu comprovar as demais. O autor atribui este resultado a fatores humanos e organizacionais não considerados nas leis.

iii) BENDIFALLAH & SCACCHI (1987):

- apresentam dados qualitativos e análises comparativas de dois estudos de casos;
- revelam que tipos similares de sistemas de software em tipos similares de características organizacionais têm trajetórias evolucionárias diferentes.

iv) TAMAI & TORIMITSU (1992):

- utilizaram um survey em organizações japonesas para avaliar software na área empresarial, por 5 anos;
- relatam que softwares de pequena escala tendem a ter vida curta;
- relatam também que aplicações administrativas (pessoal, contabilidade) tendem a ter vida mais longa que aplicações na área de negócios (suporte de vendas, manufatura).

v) COOK & ROESCH (1994):

- examinaram 10 versões por 18 meses de um sistema de tempo real na área de telefonia;
- o foco do trabalho foi na exploração de métricas de software para complexidade;
- métricas de informação seriam melhores que outras métricas, como as métricas de Halstead e McCabe, e linhas de código;
- relataram suporte às leis de evolução de software.

vi) GEFEN & SCHNEBERGER (1996):

- exploraram dois padrões distintos de modificações em manutenção de software (constante e declínio) para determinar se a distribuição de manutenção de software é homogênea;
- estudaram relatórios de problemas de software caracterizados pelo tipo de modificação (corretiva ou adaptativa), além do número de novas aplicações;
- relataram que a taxa de manutenção decresce ao longo do tempo no total, mas não se vistas em fases individuais, descritas como estabilização, melhoria e expansão.

vii) BASILI et al. (1996):

- em um estudo mais recente, examinaram 25 releases de 10 diferentes sistemas na NASA, incluindo mais de 100 sistemas de software, totalizando cerca de 4,5 milhões de linhas de código fonte, por 18 meses;
- o foco do estudo foi caracterizar tipos de atividades de manutenção e examinar o esforço total e a distribuição do esforço entre estes projetos de manutenção;
- o estudo observou três tipos de atividades de manutenção (corretiva, adaptativa e perfectiva) e um conjunto de atividades de manutenção para cada tipo;
- relataram que esforço para correção de erros, tipicamente pequenas mudanças, requerem significativa atividade isolada, enquanto melhorias requerem mais tempo em inspeção e certificação;
- relataram ainda que esforço para projeto, codificação e teste de unidade foram similares.

viii) LEHMAN et al. (1997):

- análise experimental, por 8 anos, em sistema da área financeira, com um total de 100 releases (21 analisados);

- para cada release, foi registrado o tamanho em termos do número de módulos e o número de módulos modificados;
- determinação do modelo de crescimento para o tamanho do módulo, por release;
- relatam que o padrão observado de crescimento do sistema estabilizou a partir do sexto release;
- atualizam as leis de evolução, totalizando 8 leis.

ix) KEMERER & SLAUGHTER (1997):

- examinaram padrões de atividades de manutenção em 621 módulos de software em 5 diferentes sistemas;
- relatam que módulos em softwares estratégicos são atualizados mais freqüentemente que em não-estratégicos;
- relatam também que software gerado por CASE será menos reparado que software codificado manualmente;
- relatam ainda que software com alta complexidade será mais reparado;
- afirmam que software mais antigo será atualizado e reparado mais freqüentemente que software mais novo;
- afirmam também que softwares maiores serão atualizados, reparados e terão mais manutenção preventiva que softwares menores;
- verificaram que 80% dos defeitos estão em 20% do código.

x) GALL et al. (1997):

- apresenta dados e observações baseadas em histórico de releases de um produto de software de um grande sistema na área de telecomunicações;
- o crescimento deste sistema em 20 releases é compatível com a tendência encontrada nos dados de Lehman;
- reportam que a evolução global do sistema segue as tendências e confirma as leis de evolução de software, o mesmo não acontecendo para subsistemas e módulos individuais.

xi) EICK et al. (1999):

- estudos em 100 milhões de linhas de código em C/C++ organizados em 50 subsistemas e 5 mil módulos, desenvolvido por 15 anos com a contribuição de mais de 10 mil desenvolvedores, na área de telefonia;
- definem causas para o decaimento de software;
- identificam sintomas de decaimento de software;
- identificam fatores de risco para decaimento de software;
- estabelecem índices de decaimento de software, apresentando a formulação matemática utilizada;
- apresentam evidências para o decaimento de software, apresentando a abordagem estatística utilizada;
- descrevem ainda modelos de esforço envolvidos no decaimento de software.

xii) KEMERER & SLAUGHTER (1999):

- apresentam um conjunto de dados, análises e comparações com estudos anteriores;
- análises feitas em código fonte COBOL de dois sistemas da área empresarial;
- refinaram as atividades de manutenção (corretiva, adaptativa e perfectiva) em 30 subcategorias;

- apresentam uma estratégia para classificação dos tipos de manutenção registradas no código fonte em diferentes categorias;
- apresentam, ainda, os dados coletados dos sistemas em estudo, além da abordagem estatística utilizada.

xiii) CUSUMANO & YOFFIE (1999):

- apresentam resultados de estudos de casos na Microsoft e Netscape;
- indicam forte confiança em releases incrementais de versões alfa e beta para os clientes como estratégia de negócio para melhorar a evolução das características de sistemas, de encontro às exigências do usuário;
- a satisfação do usuário pode melhorar e ser dirigida por um curto intervalo de tempo entre releases;
- afirmam que o estudo não confirma e nem refuta as leis de evolução de software, mas introduz uma nova dinâmica na evolução de software por fazer a atividade de liberação de releases uma variável independente, em vez de uma variável de controle.

xiv) PERRY et al. (2001):

- reportam os resultados de um estudo de caso de observação do desenvolvimento de grandes sistemas na área de telecomunicações;
- indicam extensivas mudanças em paralelo sendo feita entre os releases do sistema de software;
- afirmam que a noção de mudanças em paralelo pode ser um fator de confusão em atividades de manutenção e não são explicitamente apontadas pelas leis de evolução de software;
- introduzem assim, um outro fator organizacional que pode afetar a evolução de software.

xv) SCACCHI (2003):

- faz uma análise das leis de evolução no contexto de software livre;
- descreve cinco entidades para identificação dos participantes apropriados para examinar e explorar evolução de software (Evolução sobre Releases, Sistema ou Programa, Aplicação, Processo, Modelos de Processos);
- faz uma leitura das leis de evolução de software no contexto de software livre;
- afirma que as leis de evolução devem ser revistas, adaptadas ou reavaliadas para o contexto de software livre.

xvi) CAPILUPPI, LAGO & MORISIO (2003):

- apresentam análises de 12 projetos de software livre baseados em vários atributos (tamanho, módulos, desenvolvedores, produtividade, versões);
- 4 projetos foram analisados mais profundamente do ponto de vista de evolução;
- apresentam algumas estatísticas para analisar a evolução, comparando alguns atributos entre si;
- projetos de software livre resultam, normalmente, mais releases que softwares comerciais tradicionais;
- sugerem que as observações feitas não podem ser generalizadas para todo projeto de software livre;
- relataram suporte às leis de evolução de software.

A maioria dos estudos apresentados trata da evolução de software apenas na fase de codificação de sistemas legados, alguns ainda em *batch*, normalmente na linguagem COBOL.

Nenhum dos estudos trata de paradigmas mais modernos, como orientação a objetos ou ainda de outra etapa do processo de desenvolvimento que não seja a codificação, caracterizando uma grande área de pesquisa em evolução de software.

5. As Leis de Evolução Aplicadas a Processos de Desenvolvimento de Software Orientado a Objetos

Uma vez que as abordagens encontradas na literatura tratam normalmente de evolução em código fonte de sistemas legados, alguns ainda em *batch* e, normalmente, na linguagem COBOL, este trabalho traz como principal contribuição a hipótese de que as leis de evolução de software poderiam também ser suportadas pelas diferentes etapas de um processo de desenvolvimento de software baseado no paradigma da orientação a objetos, invés de apenas à fase de codificação de sistemas legados. Para isto, sem perda de generalidade, foi utilizada uma adaptação do processo de desenvolvimento de software orientado a objetos proposto em TRAVASSOS et al. (2001), ilustrado na figura 5.1.

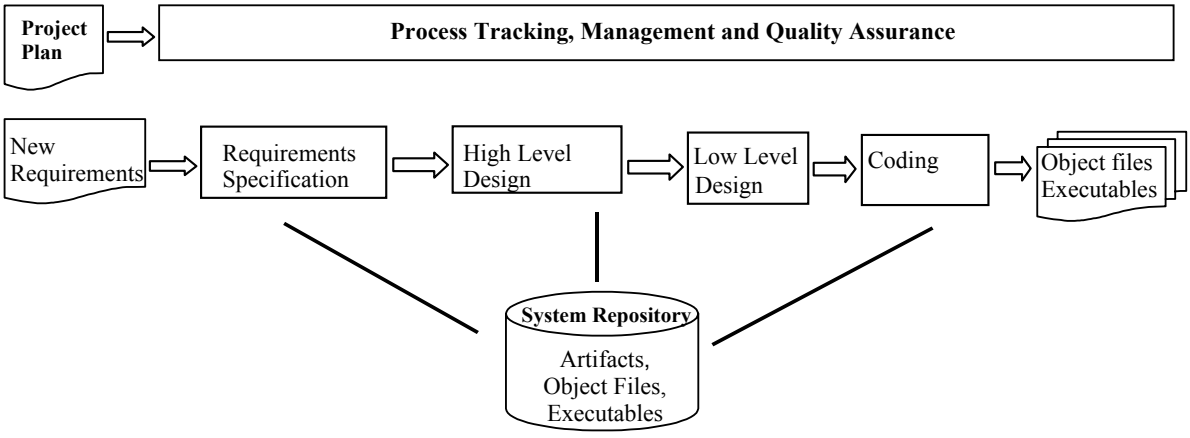

Figura 5.1 – O processo de desenvolvimento de software considerado

Desta forma, primeiramente foi identificado um conjunto de características que pudessem abranger as leis de evolução, no sentido de proporcionar um melhor entendimento do que realmente afetaria o decaimento de software. Algumas características foram adaptadas da norma ISO 9126-1 (1997) e outras foram acrescentadas no sentido de contemplar características não abrangidas por esta norma, mas importantes no processo de evolução de software.

Das características extraídas e adaptadas da ISO 9126-1, estão Confiabilidade, Eficiência e Manutenibilidade. As características acrescentadas são Tamanho, Periodicidade, Complexidade e Modularidade. Estas características são medidas através da coleta de métricas específicas a cada versão do artefato.

Caracterizamos por Tamanho a quantidade de artefatos produzidos em cada etapa do processo de desenvolvimento de software proposto, como quantidade de requisitos, diagramas de classe e linhas de código fonte, exemplificando, respectivamente, artefatos das etapas de Especificação de Requisitos, Projeto de Alto Nível e Codificação. Ainda em Tamanho, consideramos como quantidade de artefatos tratados o número de inclusões, modificações e exclusões de cada artefato. Além disso, no caso de modificações, um artefato modificado várias vezes é repetidamente contado tantas forem as modificações efetuadas.

Por Periodicidade estamos representando o intervalo de tempo decorrido entre cada versão produzida daquele artefato.

Complexidade é identificada através de elementos que possam medir a complexidade estrutural do artefato em questão, como quantidade de pontos de função para Especificação de Requisitos, quantidade de classes de domínio para Projeto de Alto Nível, quantidade de classes de suporte para Projeto de Baixo Nível e complexidade ciclomática para Codificação.

Descrevemos Modularidade através das características de acoplamento e coesão entre artefatos como, por exemplo, acoplamento entre casos de uso em Especificação de Requisitos, entre classes em Projeto de Alto e Baixo Nível e coesão em métodos em Codificação.

Por Confiabilidade representamos a quantidade de defeitos identificados por artefato em cada versão do mesmo, além da disponibilidade do sistema. Esta característica foi baseada principalmente na sub-característica de Maturidade da ISO 9126-1.

Eficiência é identificada pela quantidade de pessoas e recursos alocados, tempo consumido e produtividade média da equipe, por versão de cada artefato. Esta característica foi baseada principalmente nas sub-características de Comportamento em Relação ao Tempo e Comportamento em Relação aos Recursos da ISO 9126-1.

Por fim, Manutenibilidade é caracterizada pela eficiência na identificação de defeitos e também pela eficiência na remoção destes. Esta característica foi baseada principalmente nas sub-características de Modificabilidade e Testabilidade da ISO 9126-1.

A tabela 5.1 sumariza o relacionamento entre as Leis de Evolução de Software e as características apresentadas.

Tabela 5.1 – Leis de Evolução de Software x Características

	Tamanho	Periodicidade	Complexidade	Modularidade	Confiabilidade	Eficiência	Manutenibilidade
Mudança Contínua	✓	✓					
Incremento da Complexidade	✓		✓	✓			✓
Auto-Regulação					✓		✓
Conservação da Estabilidade Organizacional	✓					✓	
Conservação da Familiaridade	✓		✓				
Crescimento Contínuo	✓	✓					
Declínio da Qualidade				✓	✓		✓
Sistema de Realimentação	✓	✓	✓	✓	✓	✓	✓

A partir da definição das características que influenciam na evolução de software, foi feito um estudo de como estas características se comportariam para levar ao decaimento de software. Para cada Lei de Evolução, foram descritas tabelas verdade no sentido de estudar o comportamento esperado de cada uma destas características, conforme descrito a seguir. Será utilizada a seguinte simbologia para a construção das tabelas:

- ↑ aumento de determinada característica;
- ↓ diminuição de determinada característica;

- $\leftrightarrow$ uma característica se mantém constante;
- $\times$ falta de aderência à Lei de Evolução em questão;
- $\checkmark$ aderência à Lei de Evolução em questão;
- $\wedge$ conectivo lógico E;
- V contectivo lógico OU;
- $\neg$ negação;
- $\Rightarrow$ implicação lógica.

Após cada tabela, apresenta-se a hipótese de estudo associada, descrita através do formalismo definido em CARVER (2003).

Mudança Contínua

A Lei de Mudança Contínua (MC) indica que um produto em uso ou está em mudança constante ou se torna progressivamente menos útil e deve ser continuamente adaptado senão tornar-se-á progressivamente menos satisfatório.

A tabela 5.2 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Mudança Contínua, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.2 – Tabela Verdade para a Lei de Mudança Contínua

Tamanho	Periodicidade	Mudança Contínua
$\uparrow$	$\uparrow$	$\times$
$\uparrow$	$\leftrightarrow$	$\checkmark$
$\uparrow$	$\downarrow$	$\checkmark$
$\leftrightarrow$	$\uparrow$	$\times$
$\leftrightarrow$	$\leftrightarrow$	$\checkmark$
$\leftrightarrow$	$\downarrow$	$\checkmark$
$\downarrow$	$\uparrow$	$\times$
$\downarrow$	$\leftrightarrow$	$\times$
$\downarrow$	$\downarrow$	$\times$
$\neg\downarrow$	$\wedge$ $\neg\uparrow$	$\Rightarrow$ **MC**

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Tamanho não diminui $\wedge$ Periodicidade não aumenta) $\Rightarrow$ Mudança Contínua

Incremento da Complexidade

A Lei de Incremento da Complexidade (IC) indica que a mudança constante continuamente introduz complexidade no produto, deteriorando a estrutura do mesmo e, se não for desenvolvida nenhuma atividade explícita do controle da complexidade, a manutenção do produto deixa de ser possível e este se torna inútil.

A tabela 5.3 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Incremento da Complexidade, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.3 – Tabela Verdade para a Lei de Incremento da Complexidade

Tamanho	Complexidade	Modularidade	Manutenibilidade	Incremento da Complexidade
↑	↑	↑	↑	✓
↑	↑	↑	↔	✓
↑	↑	↑	↓	✓
↑	↑	↔	↑	✓
↑	↑	↔	↔	✓
↑	↑	↔	↓	✓
↑	↑	↓	↑	✓
↑	↑	↓	↔	✓
↑	↑	↓	↓	✓
↑	↔	↑	↑	✓
↑	↔	↑	↔	✓
↑	↔	↑	↓	✓
↑	↔	↔	↑	✓
↑	↔	↔	↔	✓
↑	↔	↔	↓	✓
↑	↔	↓	↑	✓
↑	↔	↓	↔	✓
↑	↔	↓	↓	✓
↑	↓	↑	↑	✓
↑	↓	↑	↔	✓
↑	↓	↑	↓	✓
↑	↓	↔	↑	✓
↑	↓	↔	↔	✓
↑	↓	↔	↓	✓
↑	↓	↓	↑	✓
↑	↓	↓	↔	✓
↑	↓	↓	↓	✓
↔	↑	↑	↑	✓
↔	↑	↑	↔	✓
↔	↑	↑	↓	✓
↔	↑	↔	↑	✓
↔	↑	↔	↔	✓
↔	↑	↔	↓	✓
↔	↑	↓	↑	✓
↔	↑	↓	↔	✓
↔	↑	↓	↓	✓
↔	↔	↑	↑	×
↔	↔	↑	↔	×
↔	↔	↑	↓	✓
↔	↔	↔	↑	×
↔	↔	↔	↔	×
↔	↔	↔	↓	✓
↔	↔	↓	↑	✓
↔	↔	↓	↔	✓
↔	↔	↓	↓	✓
↔	↓	↑	↑	×
↔	↓	↑	↔	×
↔	↓	↑	↓	✓
↔	↓	↔	↑	×
↔	↓	↔	↔	×
↔	↓	↔	↓	✓
↔	↓	↓	↑	✓
↔	↓	↓	↔	✓
↔	↓	↓	↓	✓
↓	↑	↑	↑	✓
↓	↑	↑	↔	✓
↓	↑	↑	↓	✓
↓	↑	↔	↑	✓
↓	↑	↔	↔	✓
↓	↑	↔	↓	✓
↓	↑	↓	↑	✓
↓	↑	↓	↔	✓
↓	↑	↓	↓	✓
↓	↔	↑	↑	×

Tamanho	Complexidade	Modularidade	Manutenibilidade	Incremento da Complexidade
↓	↔	↑	↔	×
↓	↔	↑	↓	✓
↓	↔	↔	↑	×
↓	↔	↔	↔	×
↓	↔	↔	↓	✓
↓	↔	↓	↑	✓
↓	↔	↓	↔	✓
↓	↔	↓	↓	✓
↓	↓	↑	↑	×
↓	↓	↑	↔	×
↓	↓	↑	↓	✓
↓	↓	↔	↑	×
↓	↓	↔	↔	×
↓	↓	↔	↓	✓
↓	↓	↓	↑	✓
↓	↓	↓	↔	✓
↓	↓	↓	↓	✓
↑	v	↑	v	↓	v	↓	⇒	IC

Assim, apresentamos a seguinte hipótese para estudo desta lei:
(Tamanho aumenta **V** Complexidade aumenta **V** Modularidade diminui **V** Manutenibilidade diminui) ⇒ Incremento da Complexidade

Auto-Regulação

A Lei de Auto-Regulação (AR) indica que o processo de evolução do produto é uma dinâmica auto regulada com tendências estatísticas determináveis e invariâncias e que sistemas de software exibem comportamentos regulares e tendências que podem ser medidas e previstas.

A tabela 5.4 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Auto-Regulação, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.4 – Tabela Verdade para a Lei de Auto-Regulação

Confiabilidade	Manutenibilidade	Auto-Regulação
↑	↑	✓
↑	↔	✓
↑	↓	×
↔	↑	✓
↔	↔	✓
↔	↓	×
↓	↑	×
↓	↔	×
↓	↓	×
¬↓	^	¬↓	⇒	AR

Assim, apresentamos a seguinte hipótese para estudo desta lei:
(Confiabilidade não diminui **Λ** Manutenibilidade não diminui) ⇒ Auto-Regulação

Conservação da Estabilidade Organizacional

A Lei de Conservação da Estabilidade Organizacional (CEO) indica que a taxa de atividade global em um produto em evolução é estatisticamente invariante ao longo de seu ciclo de vida; que atributos organizacionais, como produtividade, não exibem grandes

flutuações e que recursos e resultados alcançam um nível ótimo, e adicionar mais recursos não muda significativamente os resultados.

A tabela 5.5 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Conservação da Estabilidade Organizacional, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.5 – Tabela Verdade para a Lei de Conservação da Estabilidade Organizacional

Tamanho		Eficiência		Conservação da Estabilidade Organizacional
↑		↑		×
↑		↔		×
↑		↓		×
↔		↑		×
↔		↔		✓
↔		↓		×
↓		↑		×
↓		↔		×
↓		↓		×
↔	^	↔	⇒	**CEO**

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Tamanho constante Λ Eficiência constante) ⇒ Conservação da Estabilidade Organizacional

Conservação da Familiaridade

A Lei de Conservação da Familiaridade (CF) indica que, durante o ciclo de vida de um produto, o conteúdo de cada versão é estatisticamente invariante; que o conteúdo de sucessivas versões de programas em evolução (mudanças, adições, exclusões) é estatisticamente invariante, e; após um tempo, o efeito de versões de manutenções sucessivas faz pouca diferença na funcionalidade geral.

A tabela 5.6 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Conservação da Familiaridade, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.6 – Tabela Verdade para a Lei de Conservação da Familiaridade

Tamanho		Complexidade		Conservação da Familiaridade
↑		↑		×
↑		↔		×
↑		↓		×
↔		↑		×
↔		↔		✓
↔		↓		×
↓		↑		×
↓		↔		×
↓		↓		×
↔	^	↔	⇒	**CF**

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Tamanho constante Λ Complexidade constante) ⇒ Conservação da Familiaridade

Crescimento Contínuo

A Lei de Crescimento Contínuo (CC) indica que o conteúdo funcional de um sistema deve ser continuamente incrementado para manter a satisfação do usuário ao longo do ciclo de vida.

A tabela 5.7 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Crescimento Contínuo, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.7 – Tabela Verdade para a Lei de Crescimento Contínuo

Tamanho	Periodicidade	Crescimento Contínuo
↑	↑	×
↑	↔	✓
↑	↓	✓
↔	↑	×
↔	↔	×
↔	↓	×
↓	↑	×
↓	↔	×
↓	↓	×
↑	∧	¬↑ ⇒ CC

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Tamanho aumenta ∧ Periodicidade não aumenta) ⇒ Crescimento Contínuo

Declínio da Qualidade

A Lei de Declínio da Qualidade (DQ) indica que a qualidade de um sistema entrará em declínio a menos que seja rigorosamente mantida e adaptada às mudanças do ambiente operacional.

A tabela 5.8 apresenta o estudo do impacto de cada característica definida para o estudo da Lei de Declínio da Qualidade, apresentando ao final a formulação lógica que determina como estas características se comportam para determinar logicamente esta Lei de Evolução.

Tabela 5.8 – Tabela Verdade para a Lei de Declínio da Qualidade

Modularidade	Confiabilidade	Manutenibilidade	Declínio da Qualidade
↑	↑	↑	×
↑	↑	↔	×
↑	↑	↓	✓
↑	↔	↑	×
↑	↔	↔	×
↑	↔	↓	✓
↑	↓	↑	✓
↑	↓	↔	✓
↑	↓	↓	✓
↔	↑	↑	×
↔	↑	↔	×
↔	↑	↓	✓
↔	↔	↑	×
↔	↔	↔	×
↔	↔	↓	✓
↔	↓	↑	✓
↔	↓	↔	✓
↔	↓	↓	✓

Modularidade	Confiabilidade	Manutenibilidade	Declínio da Qualidade
↓	↑	↑	✓
↓	↑	↔	✓
↓	↑	↓	✓
↓	↔	↑	✓
↓	↔	↔	✓
↓	↔	↓	✓
↓	↓	↑	✓
↓	↓	↔	✓
↓	↓	↓	✓
↓ v	↓ v	↓	⇒ DQ

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Modularidade diminui **V** Confiabilidade diminui **V** Manutenibilidade diminui) ⇒ Declínio da Qualidade

Sistema de Realimentação

A Lei de Sistema de Realimentação (SR) indica que 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 a alcançar significativas melhorias.

Interpretamos esta lei como resultado das análises das leis anteriores e, portanto, não apresenta uma tabela verdade específica para sua definição lógica.

Assim, apresentamos a seguinte hipótese para estudo desta lei:

(Coleta das medidas relativas a tamanho, periodicidade, complexidade, modularidade, confiabilidade, eficiência, manutenibilidade) ⇒ Sistema de Realimentação

Sumarizando as hipóteses, temos:

(Tamanho não diminui **Λ** Periodicidade não aumenta) ⇒ Mudança Contínua

(Tamanho aumenta **V** Complexidade aumenta **V** Modularidade diminui **V** Manutenibilidade diminui) ⇒ Incremento da Complexidade

(Confiabilidade não diminui **Λ** Manutenibilidade não diminui) ⇒ Auto-Regulação

(Tamanho constante **Λ** Eficiência constante) ⇒ Conservação da Estabilidade Organizacional

(Tamanho constante **Λ** Complexidade constante) ⇒ Conservação da Familiaridade

(Tamanho aumenta **Λ** Periodicidade não aumenta) ⇒ Crescimento Contínuo

(Modularidade diminui **V** Confiabilidade diminui **V** Manutenibilidade diminui) ⇒ Declínio da Qualidade

(Coleta das medidas relativas a tamanho, periodicidade, complexidade, modularidade, confiabilidade, eficiência, manutenibilidade) ⇒ Sistema de Realimentação

5.1 Mapeando as hipóteses ao longo do processo através de métricas do software OO

A partir deste conjunto de hipóteses e, considerando o processo de desenvolvimento de software utilizando o paradigma da orientação a objetos proposto anteriormente, definimos um conjunto de métricas que seriam aplicáveis a cada etapa do processo de desenvolvimento.

Este conjunto de métricas define um *framework* parametrizável, permitindo flexibilidade para se coletar seus valores e estudar o decaimento de software dentro de uma das etapas do processo de desenvolvimento. Esta abordagem torna mais ameno o estudo da evolução de software, uma vez que pode ser adaptada em função do processo de desenvolvimento utilizado, coletando-se apenas aquelas métricas que puderem ser efetivamente extraídas do processo em uso. Esta flexibilidade ficará mais evidente quando associarmos as hipóteses identificadas com as etapas do processo utilizado e as métricas definidas. Este mapeamento será feito posteriormente neste trabalho e indicará o relacionamento entre as métricas e sua opcionalidade.

As métricas que dão apoio a cada uma das características descritas anteriormente nas diferentes etapas do processo de desenvolvimento foram baseadas, genericamente, nos trabalhos de PFLEEGER (2004) e PRESSMAN (2002). Métricas mais específicas relativas ao contexto do paradigma da orientação a objetos foram baseadas em CHIDAMBER & KEMERER (1994), LORENZ & KIDD (1994), TRAVASSOS et al. (2001) e TRAVASSOS (2003).

O conjunto de métricas associadas a cada característica em cada etapa do processo de desenvolvimento está representado na tabela 5.9.

Tabela 5.9 - Métricas associadas por Característica em cada etapa do Processo

	Tamanho	Periodicidade	Complexidade	Modularidade	Confiabilidade	Eficiência	Manutenibilidade
Especificação de Requisitos	• Qtde Requisitos • Qtde Casos de Uso • Qtde Requisitos Tratados • Qtde Casos de Uso Tratados	• Intervalo entre Versões	• Qtde Pontos de Função • Qtde Pontos de Casos de Uso	• Acoplamento entre Casos de Uso (Qtde Extensões e Usos)	• Qtde Defeitos	• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe	• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos
Projeto de Alto Nível	• Qtde Diagramas Classes • Qtde Diagramas Sequência • Qtde Diagramas Estado • Qtde Diagramas Empacotamento • Qtde Diagramas Atividades • Qtde Diagramas Classes Tratados • Qtde Diagramas Sequência Tratados • Qtde Diagramas Estado Tratados • Qtde Diagramas Empacotamento Tratados • Qtde Diagramas Atividades Tratados	• Intervalo entre Versões	• Qtde Classes • Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe	• Acoplamento entre Classes	• Qtde Defeitos	• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe	• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos
Projeto de Baixo Nível	• Qtde Diagramas Classe • Qtde Diagramas Sequência • Qtde Diagramas Classe Tratados • Qtde Diagramas Sequência Tratados	• Intervalo entre Versões	• Qtde Classes de Domínio • Qtde Classes de Suporte • Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe • Acoplamento entre Objetos • Resposta de uma Classe • Perda de Coesão em Métodos • Qtde Subsistemas	• Coesão em Métodos • Acoplamento entre Classes	• Qtde Defeitos	• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe	• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos
Codificação	• Qtde Linhas de Código Fonte • Qtde Linhas de Código Fonte Tratadas	• Intervalo entre Versões	• Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe • Acoplamento entre Objetos • Resposta de uma Classe • Perda de Coesão em Métodos • Qtde Subsistemas • Complexidade Ciclomática	• Coesão em Métodos • Acoplamento entre Classes	• Qtde Defeitos • Disponibilidade do Sistema	• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe	• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos

A partir deste conjunto de métricas, pôde-se fazer um estudo no sentido de verificar o comportamento destas métricas para as hipóteses anteriormente apresentadas.

Para cada etapa do processo de desenvolvimento de software, foi elaborada uma tabela relacionando as métricas identificadas na tabela 5.9 através da interpretação das hipóteses apresentadas, ou seja, relacionando métricas com as Leis de Evolução, Características e Relacionamentos entre as Características, para cada etapa do processo. Neste contexto, a tabela 5.10 apresenta a interpretação das métricas na etapa de Especificação de Requisitos por Característica e Lei de Evolução, enquanto que as tabelas 5.11, 5.12 e 5.13 apresentam as interpretações das métricas para as etapas de Projeto de Alto Nível, Projeto de Baixo Nível e Codificação, respectivamente.

Estas tabelas apresentam, entre as colunas que identificam as Características, conectivos lógicos que mapeiam as hipóteses. Dentro de cada célula das tabelas, são apresentadas as métricas, com suas respectivas interpretações relacionadas com as hipóteses. Conectivos lógicos entre estas métricas indicam a obrigatoriedade ou não da coleta. O conectivo lógico $\vee$ (OU) entre as métricas indica que são medidas opcionais de serem coletadas, onde qualquer uma delas seria suficiente para analisar aquela característica para a Lei de Evolução em questão. Isto torna o *framework* mais flexível, deixando explícito que nem todas as métricas precisam ser coletadas, tornando-o adaptável ao processo de desenvolvimento de software que estiver sendo utilizado.

As métricas apresentadas englobam métricas de processo e produto e devem ser coletadas a cada versão produzida do artefato, compondo uma base histórica de dados de forma a podermos estudar seus efeitos no decaimento de software. Algumas métricas têm caráter subjetivo e dependem de uma *baseline* para sua interpretação.

Tabela 5.10 – Interpretação das Métricas na Etapa de Especificação de Requisitos por Característica e Lei de Evolução

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
Mudança Contínua	¬↓ Qtde Requisitos Tratados ∨ ¬↓ Qtde Casos de Uso Tratados	∧	¬↑ Intervalo entre Versões										
Incremento da Complexidade	↑ Qtde Requisitos ∨ ↑ Qtde Casos de Uso			∨	↑ Qtde Pontos de Função ∨ ↑ Qtde Pontos de Caso de Uso	∨	↑ Acoplamento entre Casos de Uso (Qtde Extensões e Usos)					∨	↓ Eficiência no Diagnóstico de Defeitos
Auto-Regulação									¬↑ Qtde Defeitos			∧	¬↓ Eficiência na Remoção de Defeitos
Conservação da Estabilidade Organizacional	↔ Qtde Requisitos Tratados ∨ ↔ Qtde Casos de Uso Tratados									∧	↔ Qtde Pessoas ∧ ↔ Recursos Alocados ∧ ↔ Tempo Consumido ∧ ↔ Produtividade Média da Equipe		
Conservação da Familiaridade	↔ Qtde Requisitos Tratados ∨ ↔ Qtde Casos de Uso Tratados			∧	↔ Qtde Pontos de Função ∨ ↔ Qtde Pontos de Casos de Uso								
Crescimento Contínuo	↑ Qtde Requisitos ∨ ↑ Qtde Casos de Uso	∧	¬↑ Intervalo entre Versões										
Declínio da Qualidade							↑ Acoplamento entre Casos de Uso (Qtde Extensões e Usos)	∨	↑ Qtde Defeitos			∨	↓ Eficiência na Remoção de Defeitos
Sistema de Realimentação	• Qtde Requisitos • Qtde Casos de Uso • Qtde Requisitos Tratados • Qtde Casos de Uso Tratados		• Intervalo entre Versões		• Qtde Pontos de Função • Qtde Pontos de Casos de Uso		• Acoplamento entre Casos de Uso (Qtde Extensões e Usos)		• Qtde Defeitos		• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe		• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos

Tabela 5.11 – Interpretação das Métricas na Etapa de Projeto de Alto Nível por Característica e Lei de Evolução

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
Mudança Contínua	¬↓ Qtde Diagramas Classes Tratados ∨ ¬↓ Qtde Diagramas Sequência Tratados ∨ ¬↓ Qtde Diagramas Estado Tratados ∨ ¬↓ Qtde Diagramas Empacotamento Tratados ∨ ¬↓ Qtde Diagramas Atividades Tratados	∧	¬↑ Intervalo entre Versões										
Incremento da Complexidade	↑ Qtde Diagramas Classes ∨ ↑ Qtde Diagramas Sequência ∨ ↑ Qtde Diagramas Estado ∨ ↑ Qtde Diagramas Empacotamento ∨ ↑ Qtde Diagramas Atividades			∨	↑ Qtde Classes ∨ ↑ Qtde Métodos por Classe ∨ ↑ Profundidade de Herança por Classe ∨ ↑ Qtde Filhos por Classe	∨	↑ Acoplamento entre Classes					∨	↓ Eficiência no Diagnóstico de Defeitos
Auto-Regulação									¬↑Qtde Defeitos			∧	¬↓ Eficiência na Remoção de Defeitos
Conservação da Estabilidade Organizacional	↔ Qtde Diagramas Classes Tratados ∨ ↔ Qtde Diagramas Sequência Tratados ∨ ↔ Qtde Diagramas Estado Tratados ∨ ↔ Qtde Diagramas Empacotamento Tratados ∨ ↔ Qtde Diagramas Atividades Tratados									∧	↔ Qtde Pessoas ∧ ↔ Recursos Alocados ∧ ↔ Tempo Consumido ∧ ↔ Produtividade Média da Equipe		
Conservação da Familiaridade	↔ Qtde Diagramas Classes Tratados ∨			∧	↔ Qtde Classes ∨ ↔ Qtde Métodos								

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
	↔ Qtde Diagramas Sequência Tratados ∨ ↔ Qtde Diagramas Estado Tratados ∨ ↔ Qtde Diagramas Empacota-mento Tratados ∨ ↔ Qtde Diagramas Atividades Tratados				por Classe ∨ ↔ Profundidade de Herança por Classe ∨ ↔ Qtde Filhos por Classe								
Crescimento Contínuo	↑ Qtde Diagramas Classes ∨ ↑ Qtde Diagramas Sequência ∨ ↑ Qtde Diagramas Estado ∨ ↑ Qtde Diagramas Empacotamento ∨ ↑ Qtde Diagramas Atividades	∧	¬↑ Intervalo entre Versões										
Declínio da Qualidade							↑ Acoplamento entre Classes	∨	↑ Qtde Defeitos			∨	↓ Eficiência na Remoção de Defeitos
Sistema de Realimentação	• Qtde Diagramas Classes • Qtde Diagramas Sequência • Qtde Diagramas Estado • Qtde Diagramas Empacotamento • Qtde Diagramas Atividades • Qtde Diagramas Classes Tratados • Qtde Diagramas Sequência Tratados • Qtde Diagramas Estado Tratados • Qtde Diagramas Empacotamento Tratados • Qtde Diagramas Atividades Tratados		• Intervalo entre Versões		• Qtde Classes • Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe		• Acoplamento entre Classes		• Qtde Defeitos		• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe		• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos

Tabela 5.12 – Interpretação das Métricas na Etapa de Projeto de Baixo Nível por Característica e Lei de Evolução

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
Mudança Contínua	¬↓ Qtde Diagramas Classe Tratados ∨ ¬↓ Qtde Diagramas Sequência Tratados	∧	¬↑ Intervalo entre Versões										
Incremento da Complexidade	↑ Qtde Diagramas Classe ∨ ↑ Qtde Diagramas Sequência			∨	↑ Qtde Classes de Domínio ∨ ↑ Qtde Classes de Suporte ∨ ↑ Qtde Métodos por Classe ∨ ↑ Profundidade de Herança por Classe ∨ ↑ Qtde Filhos por Classe ∨ ↑ Acoplamento entre Objetos ∨ ↑ Resposta de uma Classe ∨ ↑ Perda de Coesão em Métodos ∨ ↑ Qtde Subsistemas	∨	↓ Coesão em Métodos ∨ ↑ Acoplamento entre Classes					∨	↓ Eficiência no Diagnóstico de Defeitos
Auto-Regulação									¬↑Qtde Defeitos			∧	¬↓ Eficiência na Remoção de Defeitos
Conservação da Estabilidade Organizacional	↔ Qtde Diagramas Classe Tratados ∨ ↔ Qtde Diagramas Sequência Tratados									∧	↔ Qtde Pessoas ∧ ↔ Recursos Alocados ∧ ↔ Tempo Consumido ∧ ↔ Produtividade Média da Equipe		
Conservação da Familiaridade	↔ Qtde Diagramas Classe Tratados ∨ ↔ Qtde Diagramas Sequência Tratados			∧	↔ Qtde Classes de Domínio ∨ ↔ Qtde Classes de Suporte								

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
					∨ ↔ Qtde Métodos por Classe ∨ ↔ Profundidade de Herança por Classe ∨ ↔ Qtde Filhos por Classe ∨ ↔Acoplamento entre Objetos ∨ ↔ Resposta de uma Classe ∨ ↔ Perda de Coesão em Métodos ∨ ↔Qtde Subsistemas								
Crescimento Contínuo	↑ Qtde Diagramas Classe ∨ ↑ Qtde Diagramas Sequência	∧	¬↑ Intervalo entre Versões										
Declínio da Qualidade							↓ Coesão em Métodos ∨ ↑ Acoplamento entre Classes	∨	↑ Qtde Defeitos			∨	↓ Eficiência na Remoção de Defeitos
Sistema de Realimentação	• Qtde Diagramas Classe • Qtde Diagramas Sequência • Qtde Diagramas Classe Tratados • Qtde Diagramas Sequência Tratados		• Intervalo entre Versões		• Qtde Classes de Domínio • Qtde Classes de Suporte • Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe • Acoplamento entre Objetos • Resposta de uma Classe • Perda de Coesão em Métodos • Qtde Subsistemas		• Coesão em Métodos • Acoplamento entre Classes		• Qtde Defeitos		• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe		• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos

Tabela 5.13 – Interpretação das Métricas na Etapa de Codificação por Característica e Lei de Evolução

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
Mudança Contínua	¬↓ Qtde Linhas de Código Fonte Tratadas	∧	¬↑ Intervalo entre Versões										
Incremento da Complexidade	↑ Qtde Linhas de Código Fonte			∨	↑ Qtde Métodos por Classe ∨ ↑ Profundidade de Herança por Classe ∨ ↑ Qtde Filhos por Classe ∨ ↑ Acoplamento entre Objetos ∨ ↑ Resposta de uma Classe ∨ ↑ Perda de Coesão em Métodos ∨ ↑ Qtde Subsistemas ∨ ↑ Complexidade Ciclomática	∨	↓ Coesão em Métodos ∨ ↑ Acoplamento entre Classes				∨	↓ Eficiência no Diagnóstico de Defeitos
Auto-Regulação									¬↑Qtde Defeitos ∨ ¬↓Disponibilidade do Sistema			∧	¬↓ Eficiência na Remoção de Defeitos
Conservação da Estabilidade Organizacional	↔ Qtde Linhas de Código Fonte Tratadas									∧	↔ Qtde Pessoas ∧ ↔ Recursos Alocados ∧ ↔ Tempo Consumido ∧ ↔ Produtividade Média da Equipe		
Conservação da Familiaridade	↔ Qtde Linhas de Código Fonte Tratadas			∧	↔ Qtde Métodos por Classe ∨ ↔ Profundidade de Herança por Classe ∨ ↔ Qtde Filhos por Classe ∨								

	Tamanho		Periodicidade		Complexidade		Modularidade		Confiabilidade		Eficiência		Manutenibilidade
					↔Acoplamento entre Objetos ∨ ↔ Resposta de uma Classe ∨ ↔ Perda de Coesão em Métodos ∨ ↔Qtde Subsistemas ∨ ↔ Complexidade Ciclomática								
Crescimento Contínuo	↑ Qtde Linhas de Código Fonte	∧	¬↑ Intervalo entre Versões										
Declínio da Qualidade							↓ Coesão em Métodos ∨ ↑ Acoplamento entre Classes	∨	↑ Qtde Defeitos ∨ ↓ Disponibilidade do Sistema			∨	↓ Eficiência na Remoção de Defeitos
Sistema de Realimentação	• Qtde Linhas de Código Fonte • Qtde Linhas de Código Fonte Tratadas		• Intervalo entre Versões		• Qtde Métodos por Classe • Profundidade de Herança por Classe • Qtde Filhos por Classe • Acoplamento entre Objetos • Resposta de uma Classe • Perda de Coesão em Métodos • Qtde Subsistemas • Complexidade Ciclomática		• Coesão em Métodos • Acoplamento entre Classes		• Qtde Defeitos • Disponibilidade do Sistema		• Qtde Pessoas • Recursos Alocados • Tempo Consumido • Produtividade Média da Equipe		• Eficiência no Diagnóstico de Defeitos • Eficiência na Remoção de Defeitos

6. Experimentos Relacionados a Evolução de Software Orientado a Objetos

Através do método GQM (*Goal / Question / Metric*) (BASILI & WEISS, 1984) (SOLIGEN & BERGHOUT, 1999) foram construídas as hipóteses a serem verificadas experimentalmente, baseadas nas características apresentadas para cada Lei de Evolução de Software nas diferentes etapas do processo de desenvolvimento de software utilizado.

Hipótese Principal:

As leis de evolução de software, aplicadas às etapas de um processo de desenvolvimento OO, poderiam determinar modelos de previsão de decaimento mais precisos.

Meta Principal:

Analisar o decaimento de software
Com o propósito de caracterizar
Com respeito à aplicação das Leis de Evolução de Software
Do ponto de vista do Engenheiro de Software
No contexto das etapas de Especificação de Requisitos, Projeto de Alto Nível, Projeto de Baixo Nível e Codificação existentes em processos de desenvolvimento de software baseado no paradigma orientado a objetos

Em virtude da meta principal, acima descrita, ser muito abrangente, neste trabalho consideramos a aplicação das Leis de Evolução de Software no contexto de cada uma das etapas do processo de desenvolvimento de software utilizado, estabelecendo, assim, um conjunto de quatro metas distintas, com suas respectivas questões e métricas associadas, relativas às etapas de Especificação de Requisitos, Projeto de Alto Nível, Projeto de Baixo Nível e Codificação.

Meta 1:

Analisar o decaimento de software
Com o propósito de caracterizar
Com respeito à aplicação das Leis de Evolução de Software
Do ponto de vista do Engenheiro de Software
No contexto da etapa de Especificação de Requisitos existente em processos de desenvolvimento de software baseado no paradigma orientado a objetos

Questões e Métricas Associadas (coletadas por versão):

Q1.1 – Como a Lei da Mudança Contínua é vista em termos da Etapa de Especificação de Requisitos?

M1.1.1 – Quantidade de Requisitos Tratados
M1.1.2 – Quantidade de Casos de Uso Tratados
M1.1.3 – Intervalo entre Versões

Q1.2 – Como a Lei de Incremento da Complexidade é vista em termos da Etapa de Especificação de Requisitos?

- M1.2.1 – Quantidade de Requisitos
- M1.2.2 – Quantidade de Casos de Uso
- M1.2.3 – Quantidade de Pontos de Função
- M1.2.4 – Quantidade de Pontos de Caso de Uso
- M1.2.5 - Acoplamento entre Casos de Uso (Qtde Extensões e Usos)
- M1.2.6 - Eficiência no Diagnóstico de Defeitos

Q1.3 – Como a Lei de Auto-Regulação é vista em termos da Etapa de Especificação de Requisitos?

- M1.3.1 – Quantidade de Defeitos
- M1.3.2 – Eficiência na Remoção de Defeitos

Q1.4 – Como a Lei da Conservação da Estabilidade Organizacional é vista em termos da Etapa de Especificação de Requisitos?

- M1.4.1 – Quantidade de Requisitos Tratados
- M1.4.2 – Quantidade de Casos de Uso Tratados
- M1.4.3 – Quantidade de Pessoas
- M1.4.4 – Recursos Alocados
- M1.4.5 – Tempo Consumido
- M1.4.6 – Produtividade Média da Equipe

Q1.5 – Como a Lei da Conservação da Familiaridade é vista em termos da Etapa de Especificação de Requisitos?

- M1.5.1 – Quantidade de Requisitos Tratados
- M1.5.2 – Quantidade de Casos de Uso Tratados
- M1.5.3 – Quantidade de Pontos de Função
- M1.5.4 – Quantidade de Pontos de Caso de Uso

Q1.6 – Como a Lei de Crescimento Contínuo é vista em termos da Etapa de Especificação de Requisitos?

- M1.6.1 – Quantidade de Requisitos
- M1.6.2 – Quantidade de Casos de Uso
- M1.6.3 – Intervalo entre Versões

Q1.7 – Como a Lei de Declínio da Qualidade é vista em termos da Etapa de Especificação de Requisitos?

- M1.7.1 - Acoplamento entre Casos de Uso (Qtde Extensões e Usos)
- M1.7.2 – Quantidade de Defeitos
- M1.7.3 – Eficiência na Remoção de Defeitos

Q1.8 – Como a Lei de Sistema de Realimentação é vista em termos da Etapa de Especificação de Requisitos?

- M1.8.1 – Quantidade de Requisitos
- M1.8.2 – Quantidade de Casos de Uso

M1.8.3 – Quantidade de Requisitos Tratados
M1.8.4 – Quantidade de Casos de Uso Tratados
M1.8.5 – Intervalo entre Versões
M1.8.6 – Quantidade de Pontos de Função
M1.8.7 – Quantidade de Pontos de Caso de Uso
M1.8.8 - Acoplamento entre Casos de Uso (Qtde Extensões e Usos)
M1.8.9 – Quantidade de Defeitos
M1.8.10 – Quantidade de Pessoas
M1.8.11 – Recursos Alocados
M1.8.12 – Tempo Consumido
M1.8.13 – Produtividade Média da Equipe
M1.8.14 - Eficiência no Diagnóstico de Defeitos
M1.8.15 – Eficiência na Remoção de Defeitos

Meta 2:

Analisar o decaimento de software
Com o propósito de caracterizar
Com respeito à aplicação das Leis de Evolução de Software
Do ponto de vista do Engenheiro de Software
No contexto da etapa de Projeto de Alto Nível existente em processos de desenvolvimento de software baseado no paradigma orientado a objetos

Questões e Métricas Associadas (coletadas por versão):

Q2.1 – Como a Lei da Mudança Contínua é vista em termos da Etapa de Projeto de Alto Nível?

M2.1.1 – Quantidade de Diagramas de Classes Tratados
M2.1.2 – Quantidade de Diagramas de Sequência Tratados
M2.1.3 - Quantidade de Diagramas de Estado Tratados
M2.1.4 - Quantidade de Diagramas de Empacotamento Tratados
M2.1.5 - Quantidade de Diagramas de Atividades Tratados
M2.1.6 – Intervalo entre Versões

Q2.2 – Como a Lei de Incremento da Complexidade é vista em termos da Etapa de Projeto de Alto Nível?

M2.2.1 – Quantidade de Diagramas de Classes
M2.2.2 – Quantidade de Diagramas de Sequência
M2.2.3 - Quantidade de Diagramas de Estado
M2.2.4 - Quantidade de Diagramas de Empacotamento
M2.2.5 - Quantidade de Diagramas de Atividades
M2.2.6 - Quantidade de Classes
M2.2.7 - Quantidade de Métodos por Classe
M2.2.8 - Profundidade de Herança por Classe
M2.2.9 - Quantidade de Filhos por Classe
M2.2.10 - Acoplamento entre Classes
M2.2.11 - Eficiência no Diagnóstico de Defeitos

Q2.3 – Como a Lei de Auto-Regulação é vista em termos da Etapa de Projeto de Alto Nível?

M2.3.1 – Quantidade de Defeitos
M2.3.2 – Eficiência na Remoção de Defeitos

Q2.4 – Como a Lei da Conservação da Estabilidade Organizacional é vista em termos da Etapa de Projeto de Alto Nível?

M2.1.1 – Quantidade de Diagramas de Classes Tratados
M2.1.2 – Quantidade de Diagramas de Sequência Tratados
M2.1.3 - Quantidade de Diagramas de Estado Tratados
M2.1.4 - Quantidade de Diagramas de Empacotamento Tratados
M2.1.5 - Quantidade de Diagramas Atividades Tratados
M2.4.6 – Quantidade de Pessoas
M2.4.7 – Recursos Alocados
M2.4.8 – Tempo Consumido
M2.4.9 – Produtividade Média da Equipe

Q2.5 – Como a Lei da Conservação da Familiaridade é vista em termos da Etapa de Projeto de Alto Nível?

M2.5.1 – Quantidade de Diagramas de Classes Tratados
M2.5.2 – Quantidade de Diagramas de Sequência Tratados
M2.5.3 - Quantidade de Diagramas de Estado Tratados
M2.5.4 - Quantidade de Diagramas de Empacotamento Tratados
M2.5.5 - Quantidade de Diagramas de Atividades Tratados
M2.5.6 - Quantidade de Classes
M2.5.7 - Quantidade de Métodos por Classe
M2.5.8 - Profundidade de Herança por Classe
M2.5.9 - Quantidade de Filhos por Classe

Q2.6 – Como a Lei de Crescimento Contínuo é vista em termos da Etapa de Projeto de Alto Nível?

M2.6.1 – Quantidade de Diagramas de Classes
M2.6.2 – Quantidade de Diagramas de Sequência
M2.6.3 - Quantidade de Diagramas de Estado
M2.6.4 - Quantidade de Diagramas de Empacotamento
M2.6.5 - Quantidade de Diagramas de Atividades
M2.6.6 – Intervalo entre Versões

Q2.7 – Como a Lei de Declínio da Qualidade é vista em termos da Etapa de Projeto de Alto Nível?

M2.7.1 - Acoplamento entre Classes
M2.7.2 – Quantidade de Defeitos
M2.7.3 – Eficiência na Remoção de Defeitos

Q2.8 – Como a Lei de Sistema de Realimentação é vista em termos da Etapa de Projeto de Alto Nível?

M2.8.1 – Quantidade de Diagramas de Classes

M2.8.2 – Quantidade de Diagramas de Sequência
M2.8.3 - Quantidade de Diagramas de Estado
M2.8.4 - Quantidade de Diagramas de Empacotamento
M2.8.5 - Quantidade de Diagramas de Atividades
M2.8.6 – Quantidade de Diagramas de Classes Tratados
M2.8.7 – Quantidade de Diagramas de Sequência Tratados
M2.8.8 - Quantidade de Diagramas de Estado Tratados
M2.8.9 - Quantidade de Diagramas de Empacotamento Tratados
M2.8.10 - Quantidade de Diagramas de Atividades Tratados
M2.8.11 – Intervalo entre Versões
M2.8.12 - Quantidade de Classes
M2.8.13 - Quantidade de Métodos por Classe
M2.8.14 - Profundidade de Herança por Classe
M2.8.15 - Quantidade de Filhos por Classe
M2.8.16 - Acoplamento entre Classes
M2.8.17 – Quantidade de Defeitos
M2.8.18 – Quantidade de Pessoas
M2.8.19 – Recursos Alocados
M2.8.20 – Tempo Consumido
M2.8.21 – Produtividade Média da Equipe
M2.8.22 - Eficiência no Diagnóstico de Defeitos
M2.8.23 – Eficiência na Remoção de Defeitos

Meta 3:

Analisar o decaimento de software
Com o propósito de caracterizar
Com respeito à aplicação das Leis de Evolução de Software
Do ponto de vista do Engenheiro de Software
No contexto da etapa de Projeto de Baixo Nível existente em processos de desenvolvimento de software baseado no paradigma orientado a objetos

Questões e Métricas Associadas (coletadas por versão):

Q3.1 – Como a Lei da Mudança Contínua é vista em termos da Etapa de Projeto de Baixo Nível?

M3.1.1 – Quantidade de Diagramas de Classes Tratados
M3.1.2 – Quantidade de Diagramas de Sequência Tratados
M3.1.3 – Intervalo entre Versões

Q3.2 – Como a Lei de Incremento da Complexidade é vista em termos da Etapa de Projeto de Baixo Nível?

M3.2.1 – Quantidade de Diagramas de Classes
M3.2.2 – Quantidade de Diagramas de Sequência
M3.2.3 – Quantidade de Classes de Domínio
M3.2.4 – Quantidade de Classes de Suporte
M3.2.5 – Quantidade de Métodos por Classe
M3.2.6 – Profundidade de Herança por Classe

M3.2.7 – Quantidade de Filhos por Classe
M3.2.8 – Acoplamento entre Objetos
M3.2.9 – Resposta de uma Classe
M3.2.10 – Perda de Coesão em Métodos
M3.2.11 – Quantidade de Subsistemas
M3.2.12 – Coesão em Métodos
M3.2.13 – Acoplamento entre Classes
M3.2.14 – Eficiência no Diagnóstico de Defeitos

Q3.3 – Como a Lei de Auto-Regulação é vista em termos da Etapa de Projeto de Baixo Nível?
M3.3.1 – Quantidade de Defeitos
M3.3.2 – Eficiência na Remoção de Defeitos

Q3.4 – Como a Lei da Conservação da Estabilidade Organizacional é vista em termos da Etapa de Projeto de Baixo Nível?
M3.4.1 – Quantidade de Diagramas de Classes Tratados
M3.4.2 – Quantidade de Diagramas de Sequência Tratados
M3.4.3 – Quantidade de Pessoas
M3.4.4 – Recursos Alocados
M3.4.5 – Tempo Consumido
M3.4.6 – Produtividade Média da Equipe

Q3.5 – Como a Lei da Conservação da Familiaridade é vista em termos da Etapa de Projeto de Baixo Nível?
M3.5.1 – Quantidade de Diagramas de Classes Tratados
M3.5.2 – Quantidade de Diagramas de Sequência Tratados
M3.5.3 – Quantidade de Classes de Domínio
M3.5.4 – Quantidade de Classes de Suporte
M3.5.5 – Quantidade de Métodos por Classe
M3.5.6 – Profundidade de Herança por Classe
M3.5.7 – Quantidade de Filhos por Classe
M3.5.8 – Acoplamento entre Objetos
M3.5.9 – Resposta de uma Classe
M3.5.10 – Perda de Coesão em Métodos
M3.5.11 – Quantidade de Subsistemas

Q3.6 – Como a Lei de Crescimento Contínuo é vista em termos da Etapa de Projeto de Baixo Nível?
M3.6.1 – Quantidade de Diagramas de Classes
M3.6.2 – Quantidade de Diagramas de Seqüência
M3.6.3 – Intervalo entre Versões

Q3.7 – Como a Lei de Declínio da Qualidade é vista em termos da Etapa de Projeto de Baixo Nível?
M3.7.1 – Coesão em Métodos
M3.7.2 – Acoplamento entre Classes

M3.7.3 – Quantidade de Defeitos
M3.7.4 – Eficiência na Remoção de Defeitos

Q3.8 – Como a Lei de Sistema de Realimentação é vista em termos da Etapa de Projeto de Baixo Nível?

M3.8.1 – Quantidade de Diagramas de Classes
M3.8.2 – Quantidade de Diagramas de Sequência
M3.8.3 – Quantidade de Diagramas de Classes Tratados
M3.8.4 – Quantidade de Diagramas de Sequência Tratados
M3.8.5 – Intervalo entre Versões
M3.8.6 – Quantidade de Classes de Domínio
M3.8.7 – Quantidade de Classes de Suporte
M3.8.8 – Quantidade de Métodos por Classe
M3.8.9 – Profundidade de Herança por Classe
M3.8.10 – Quantidade de Filhos por Classe
M3.8.11 – Acoplamento entre Objetos
M3.8.12 – Resposta de uma Classe
M3.8.13 – Perda de Coesão em Métodos
M3.8.14 – Quantidade de Subsistemas
M3.8.15 – Coesão em Métodos
M3.8.16 – Acoplamento entre Classes
M3.8.17 – Quantidade de Defeitos
M3.8.18 – Quantidade de Pessoas
M3.8.19 – Recursos Alocados
M3.8.20 – Tempo Consumido
M3.8.21 – Produtividade Média da Equipe
M3.8.22 – Eficiência no Diagnóstico de Defeitos
M3.8.23 – Eficiência na Remoção de Defeitos

Meta 4:

Analisar o decaimento de software
Com o propósito de caracterizar
Com respeito à aplicação das Leis de Evolução de Software
Do ponto de vista do Engenheiro de Software
No contexto da etapa Codificação existente em processos de desenvolvimento de software baseado no paradigma orientado a objetos

Questões e Métricas Associadas (coletadas por versão):

Q4.1 – Como a Lei da Mudança Contínua é vista em termos da Etapa de Codificação?

M4.1.1 – Quantidade de Linhas de Código Fonte Tratadas
M4.1.2 – Intervalo entre Versões

Q4.2 – Como a Lei de Incremento da Complexidade é vista em termos da Etapa de Codificação?

M4.2.1 – Quantidade de Linhas de Código Fonte Tratadas

M4.2.2 – Quantidade de Métodos por Classe
M4.2.3 – Profundidade de Herança por Classe
M4.2.4 – Quantidade de Filhos por Classe
M4.2.5 – Acoplamento entre Objetos
M4.2.6 – Resposta de uma Classe
M4.2.7 – Perda de Coesão em Métodos
M4.2.8 – Quantidade de Subsistemas
M4.2.9 – Complexidade Ciclomática
M4.2.10 – Coesão em Métodos
M4.2.11 – Acoplamento entre Classes
M4.2.12 – Eficiência no Diagnóstico de Defeitos

Q4.3 – Como a Lei de Auto-Regulação é vista em termos da Etapa de Codificação?
M4.3.1 – Quantidade de Defeitos
M4.3.2 – Disponibilidade do Sistema
M4.3.3 – Eficiência na Remoção de Defeitos

Q4.4 – Como a Lei da Conservação da Estabilidade Organizacional é vista em termos da Etapa de Codificação?
M4.4.1 – Quantidade de Linhas de Código Fonte Tratadas
M4.4.2 – Quantidade de Pessoas
M4.4.3 – Recursos Alocados
M4.4.4 – Tempo Consumido
M4.4.5 – Produtividade Média da Equipe

Q4.5 – Como a Lei da Conservação da Familiaridade é vista em termos da Etapa de Codificação?
M4.5.1 – Quantidade de Linhas de Código Fonte Tratadas
M4.5.2 – Quantidade de Métodos por Classe
M4.5.3 – Profundidade de Herança por Classe
M4.5.4 – Quantidade de Filhos por Classe
M4.5.5 – Acoplamento entre Objetos
M4.5.6 – Resposta de uma Classe
M4.5.7 – Perda de Coesão em Métodos
M4.5.8 – Quantidade de Subsistemas
M4.5.9 – Complexidade Ciclomática

Q4.6 – Como a Lei de Crescimento Contínuo é vista em termos da Etapa de Codificação?
M4.6.1 – Quantidade de Linhas de Código Fonte
M4.6.2 – Intervalo entre Versões

Q4.7 – Como a Lei de Declínio da Qualidade é vista em termos da Etapa de Codificação?
M4.7.1 – Coesão em Métodos
M4.7.2 – Acoplamento entre Classes

M4.7.3 – Quantidade de Defeitos
M4.7.4 – Disponibilidade do Sistema
M4.7.5 – Eficiência na Remoção de Defeitos

Q4.8 – Como a Lei de Sistema de Realimentação é vista em termos da Etapa de Codificação?

M4.8.1 – Quantidade de Linhas de Código Fonte
M4.8.1 – Quantidade de Linhas de Código Fonte Tratadas
M4.8.2 – Intervalo entre Versões
M4.8.3 – Quantidade de Métodos por Classe
M4.8.4 – Profundidade de Herança por Classe
M4.8.5 – Quantidade de Filhos por Classe
M4.8.6 – Acoplamento entre Objetos
M4.8.7 – Resposta de uma Classe
M4.8.8 – Perda de Coesão em Métodos
M4.8.9 – Quantidade de Subsistemas
M4.8.10 – Complexidade Ciclomática
M4.8.11 – Coesão em Métodos
M4.8.12 – Acoplamento entre Classes
M4.8.13 – Quantidade de Defeitos
M4.8.14 – Disponibilidade do Sistema
M4.8.15 – Quantidade de Pessoas
M4.8.16 – Recursos Alocados
M4.8.17 – Tempo Consumido
M4.8.18 – Produtividade Média da Equipe
M4.8.19 – Eficiência no Diagnóstico de Defeitos
M4.8.20 – Eficiência na Remoção de Defeitos

7. Considerações Finais e Perspectivas Futuras

Existem poucos estudos experimentais sobre evolução de software. Dentre as publicações encontradas na literatura, a maioria refere-se a estudos de observação, tratando normalmente de evolução em código fonte de sistemas legados, alguns ainda em *batch* e, normalmente, na linguagem COBOL.

Como não foi encontrada na literatura nenhum material relativo à evolução de software em codificação de sistemas utilizando o paradigma da orientação a objetos, tampouco relativo a outras etapas do processo de desenvolvimento senão a codificação, a principal contribuição deste trabalho consiste na elaboração da hipótese de que as leis de evolução de software poderiam também ser suportadas pelas diferentes etapas de um processo de desenvolvimento de software suportado pelo paradigma da orientação a objetos, ao invés de apenas à fase de codificação de sistemas legados.

O trabalho apresentou um framework conceitual de apoio à definição de um conjunto de estudos experimentais para o estudo do decaimento de software baseado nas Leis de Evolução em diferentes níveis de abstração de processos de desenvolvimento de software orientado a objetos. Um conjunto de hipóteses é apresentado, descrito através do método GQM (*Goal / Question / Metric*) (BASILI & WEISS, 1984) (SOLIGEN & BERGHOUT, 1999).

Como sugestão de trabalhos futuros, segue-se:

- Considerar outras técnicas importantes, como Reutilização de Software e Desenvolvimento Baseado em Componentes (LEHMAN & RAMIL, 2000), Famílias de Produtos (RIVA & ROSSO, 2002) e Arquiteturas de Software;
- Modificar o processo de desenvolvimento para inserir etapas de coleta e análise das métricas;
- Refinar o conjunto de hipóteses, metas, questões e métricas;
- Elaborar estudos para avaliar como as características se relacionam, ao longo do processo de desenvolvimento de software;
- Planejar e executar estudos experimentais para avaliação das hipóteses levantadas, conforme processo de experimentação definido em (AMARAL & TRAVASSOS, 2003), utilizando técnicas de simulação em dinâmica de sistemas (BARROS et al., 2004);
- Considerar aspectos de rejuvenescimento de software (redocumentação, reestruturação, engenharia reversa, reengenharia) (PFLEEGER, 2004);
- Propor e construir um ambiente que ofereça apoio ferramental para estudos experimentais em evolução de software.

Assim, esperamos poder avaliar experimentalmente a aplicabilidade das Leis de Evolução de Software no contexto de processos de desenvolvimento de software OO através de um framework que permita-nos estudar as causas de decaimento e suas conseqüências no processo de desenvolvimento de software.

Referências Bibliográficas

AMARAL & TRAVASSOS, 2003. A Package Model for Software Engineering Experiments. IEEE International Symposium on Empirical Software Engineering, 2003.

BARROS et al., 2004. Supporting Risks in Software Project Management". Journal Of Systems And Software, 2004.

BASILI & WEISS, 1984. A methodology for collecting valid software engineering data. IEEE Transactions on Software Engineering, Vol. SE-10, No. 6, 1984.

BASILI et al., 1996. Understanding and Predicting the Process of Software Maintenance Releases. Proc. 18th Int´l Conf. Software Eng., 1996.

BAUER & PIZKA, 2003. The Contribution of Free Software to Software Evolution. Sixth International Workshop on Principles of Software Evolution (IWPSE'03), IEEE, 2003.

BARRY et al, 1999. An Empirical Analysis of Software Evolution Profiles and Outcomes. Proceeding of the 20th international conference on Information Systems. ACM, 1999.

BELADY & LEHMAN, 1976. A Model of Large Program Development. IBM Systems J., vol. 15, no. 1, 1976.

BENDIFALLAH & SCACCHI, 1990. Understanding Software Maintenance Work. IEEE Trans. Software Engineering, 1990.

CAPILUPPI, 2003. Models for the Evolution of OS Projects. Proceedings of the International Conference on Software Maintenance. IEEE, 2003.

CAPILUPPI et al, 2003. Quantitative Models for Open Source Projects: A Proposal. 2003.

CAPILUPPI et al, 2003. Characteristics of Open Source Projects. Proceedings of the Seventh European Conference on Software Maintenance and Reengineering (CSMR´03). IEEE, 2003.

CARVER, 2003. The Impact of Background and Experience on Software Inspections. PhD Thesis, Faculty of the Graduate School of the University of Maryland, College Park. 2003.

CHIDAMBER & KEMERER, 1994. A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering, vol. 20, No. 6, 1994.

COOK & ROESCH, 1994. Real-Time Software Metrics. J. Systems and Software, vol. 24, no. 3, 1994.

CUSUMANO & YOFFIE, 1999. Software Development on Internet Time. Computer, October 1999.

EICK et al, 1999. Does Code Decay? Assessing the Evidence from Change Management Data. IEEE Computer, 1999.

GALL et al., 1997. Software Evolution Observations Based on Product Release History. Proc. 1997 Intern. Conf. Software Maintenance (ICSM´97), 1997.

GEFEN & SCHNEBERGER, 1996. The Non-Homogeneous Maintenance Periods: A Case Study of Software Modifications. Proc. Conf. Software Maintenance, IEEE, 1996.

GODFREY & TU, 2000. Evolution in Open Source Software: A Case Study. IEEE, 2000.

GODFREY & TU, 2001. Growth, Evolution, and Structural Change in Open Source Software. ACM, 2001.

GREENWOOD, et al, 1998. An Empirical Study of the Evolution of a Software System. Thirteenth IEEE Conference on Automated Software Engineering, 1998.

ISO 9126-1, 1997. International Standard. Information Technology – Software Quality Characteristics and Metrics – Part 1: Quality Characteristics and Sub-Characteristics, 1997.

KEMERER & SLAUGHTER, 1999, An Empirical Approach to Studying Software Evolution. IEEE, 1999.

KEMERER & SLAUGHTER, 1997. Determinants of Software Maintenance Profiles: An Empirical Investigation. Journal of Software Maintenance: Research and Practice, v.9 n.4, p.235-251, July-Aug. 1997.

LEHMAN, 1980. Programs, Life Cycle and the Laws of Software Evolution, Proc. IEEE, IEEE,1980.

LEHMAN & RAMIL, 2000. Software Evolution in the age of component-based software engineering. IEEE Software, 2000.

LEHMAN & RAMIL, 2001. Towards a Theory of Software Evolution – And its Practical Impact. IEEE, 2001.

LEHMAN & RAMIL, 2002. An Overview of Some Lessons Learnt in FEAST. WESS'02 Eighth IEEE Workshop on Empirical Studies of Software Maintenance, Canada, 2002.

LEHMAN & RAMIL, 2003. "Software Evolution – Background, Theory, Practice". Information Processing Letters, vol. 88, pp. 33 – 44, 2003.

LEHMAN, 1998. Software´s Future: Managing Evolution. IEEE Software, 1998.

LEHMAN et al, 1998. Implications of Evolution Metrics on Software Maintenance. IEEE, 1998.

LEHMAN et al, 1997. Metrics and Laws of Software Evolution – The Nineties View. IEEE, 1997.

LORENZ & KIDD, 1994. Object-Oriented Software Metrics. Prentice-Hall, 1994.

MUNSON & WERRIES, 1996. Measuring Software Evolution. IEEE, 1996.

PARNAS, 1994. Software Aging. IEEE, 1994.

PERRY, 1994. Dimensions of Software Evolution. Proceedings of the International Conference on Software Maintenance. IEEE. 1994.

PERRY et al., 2001. Parallel Changes in Large-Scale Software Development: An Observational Case Study. ACM Trans. Software Engineering and Methodology, 2001.

PFLEEGER, 1998. The Nature of System Change. IEEE Software, 1998.

PFLEEGER, 2004. Engenharia de Software – Teoria e Prática, 2a. Ed., Prentice Hall, 2004.

PRESSMAN, 2002. Engenharia de Software, 5a. Ed., Mc-Graw Hill, 2002.

RAMIL, 2002. Laws of Software Evolution and their Empirical Support. Proceedings of the International Conference on Software Maintenance (ICSM´02), IEEE, 2002.

RIVA & ROSSO, 2002. Experiences with Software Product Family Evolution. Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSE´03), IEEE, 2002.

SCACCHI, 2003. Understanding Open Source Software Evolution: Applying, Breaking, and Rethinking the Laws of Software Evolution. http://www.ics.uci.edu/~wscacchi/Papers/New/Understanding-OSS-Evolution.pdf, 2003.

SMITH & RAMIL, 2002. Qualitative Simulation of Software Evolution Process. WESS´02 Eighth Workshop on Empirical Studies of Software Maintenance, 2002.

SOLIGEN & BERGHOUT, 1999. The Goal/Question/Metric Method: a practical guide for quality improvement of software development. McGraw-Hill Publishing Company, 1999.

TAMAI & TORIMITSU, 1992. Software Lifetime and its Evolution Process over Generations. Proc. Conf. Software Maintenance, IEEE, 1992.

TRAVASSOS et al, 2001. Working with UML: A Software Design Process Based on Inspections for the Unified Modeling Language. Advances in Computers, 2001.

TRAVASSOS, 2003. Notas de aula da disciplina Tópicos Especiais em Engenharia de Software – Revisões, Inspeção e Teste de Software Orientado a Objetos. Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ, 2003.

YUEN, 1985. An Empirical Approach to the Study of Errors in Large Software under Maintenance. Proc. Second Conf. Software Maintenance, IEEE, 1985.

YUEN, 1987. A Statistical Rationale for Evolution Dynamic Concepts. Proc. Conf. Software Maintenance, IEEE,1987.

YUEN, 1988. On Analyzing Maintenance Process Data at the Global and Detailed Levels: A Case Study. Proc. Fourth Conf. Software Maintenance, IEEE, 1988.