PIBIC/CNPq/UFCG-2009

IMPALADYNAMIC: ANALISADOR DE IMPACTO DE MUDANÇAS EM CÓDIGO DE SOFTWARE

Gustavo J. De S. Santos[1], Dalton D. S. Guerrero[2]

RESUMO

Durante o processo de desenvolvimento de *software*, a fase de manutenção consome cerca de 50% ou mais do custo total do projeto. Este custo alto está associado às atividades de planejamento e implementação de mudanças que geralmente acontecem nessa fase. A Análise de Impacto de Mudanças é a área da Engenharia de *Software* responsável por identificar as possíveis consequências de uma alteração no sistema. Este trabalho teve como objetivo desenvolver uma ferramenta de suporte à análise de impacto de mudanças em código de *software* orientados a objeto. O ImpalaDynamic é a ferramenta que realiza análise de impacto estática e pondera seu resultado através do resultado da análise dinâmica. O objetivo é identificar dependências dinâmicas não encontradas pela análise estrutural do código, aumentar a precisão do resultado, auxiliando na estimativa do custo de uma mudança antes de sua implementação e assim permitir que a evolução de um *software* seja feita de forma mais controlada.

Palavras-chave: análise de impacto, evolução de software, engenharia de software

IMPALADYNAMIC: CHANGE IMPACT ANALYZER IN SOFTWARE CODE

ABSTRACT

During the software development process, the maintenance phase cosumes about 50% or more of the project's total cost. This high cost is related to the activities of planning and implementing changes that usually happen at that stage. The Change Impact Analysis is the area of Software Engineering responsible for identifying possible consequences of a change in the system. This paper aims to develop a tool that suports change impact analysis in object-oriented software code. ImpalaDynamic is the tool that performs static analysis and consider its result through dynamic analysis. Its main goal is identify dynamic dependences not found by the structural code analysis, increasing the analysis accuracy, help the cost's estimation of a change before its implemention and thus allow he evolution of a software to made in a more controlled way.

Keywords: impact analysis, software evolution, software engineering

INTRODUÇÃO

Um sistema de *software* se caracteriza por um conjunto de componentes abstratos de *software*, a exemplo de estruturas de dados e algoritmos, encapsulados na forma de procedimentos, funções e módulos, interconectados entre si, compondo a arquitetura do *software*. Evolução de *Software* é o termo usado em Engenharia de *Software* para denotar o processo de construção de um sistema e sua conseqüente e repetida atualização para se adaptar ao contexto no qual está inserido (LEHMAN, 1980).

Para atender a constante demanda por evolução, sistemas de *software* são passíveis a mudanças durante todo seu tempo de vida. Dentre as causas para o processo de atualização, estão inseridas a

[1] Aluno do Curso de Ciência da Computação, Depto. De Sistemas e Computação, UFCG, Campina Gande , PB, E-mail: gustavojss@lcc.ufcg.edu.br

[2] Ciência da Computação, Prof. Doutor, Depto. De Sistemas e Computação, UFCG, Campina Grande, PB, E-mail: dalton@dsc.ufcg.edu.br

alteração de requisitos, a correção de erros de desenho e de código, a atualização de ferramentas de suporte, tarefas de otimização, entre outras.

As atividades relacionadas à implementação de mudanças consistem em entender o comportamento do sistema em relação à mudança, implementá-la e testar o sistema modificado (BOHNER, 2002). Tais atividades são caras e consomem cerca de metade ou mais do esforço e custo total do projeto (ERLIKH, 2000). Portanto, técnicas que identificam o impacto de uma mudança em um *software* são de grande importância antes ou após sua implementação.

A Análise de Impacto de Mudanças, ou simplesmente Análise de Impacto, pode ser definida como o processo de identificação de potenciais conseqüências de uma mudança (ARNOLD & BOHNER, 1996), e tem como objetivo auxiliar na redução de custos associados à estimativa errônea dos efeitos causados por essa mudança. As informações obtidas na análise de impacto podem ser utilizadas tanto na identificação dos efeitos que uma mudança pode causar, quanto no planejamento para a realização de uma mudança.

O resultado ideal da análise de impacto é o conjunto formado apenas pelas entidades do sistema que realmente foram afetadas, situação esta que não é trivial de alcançar. Um dos problemas identificados em análise de impacto é a imprecisão do seu resultado, isto é, o quão o conjunto de impacto é próximo do conjunto de entidades realmente impactadas. Este fato pode ocorrer de duas formas:

- Identificar uma entidade como impactada quando, de fato, esta não é realmente impactada; neste caso, podemos dizer que a entidade é um falso-positivo;
- Não identificar uma entidade como impactada quando ela realmente é; neste caso, a entidade é um falso-negativo.

Este trabalho tem o objetivo de desenvolver uma ferramenta de suporte a análise de impacto de mudanças em código de *software* orientado a objeto, a fim de auxiliar na estimativa de custo de uma mudança antes de sua implementação. O ImpalaDynamic é a ferramenta que faz uso da técnica proposta na tese de Mirna Carelli (MAIA, 2009), e pretende reduzir os falso-negativos produzidos por técnicas tradicionais (i.e. estática e dinâmica), através da combinação das análises de impacto estática e dinâmica.

METODOLOGIA

Este trabalho foi realizado no Laboratório de Redes de Petri e Métodos Formais do Departamento de Sistemas e Computação da Universidade Federal de Campina Grande. A metodologia foi dividida em quatro partes:

- Revisão Bibliográfica: estudo sobre técnicas de análise de impacto semelhantes na literatura e princípios de probabilidade e estatística, incluindo avaliação sobre a aplicação dos conceitos estudados;
- Desenvolvimento da Ferramenta: implementação do protótipo seguindo a técnica proposta na tese de (MAIA, 2009);
- Experimentação: coleta de dados a serem processados pela ferramenta;
- Avaliação dos Resultados: avaliação e validação dos resultados obtidos na etapa anterior.

A Tabela 1 representa o cronograma de realização durante os meses decorridos do projeto.

Tabela 1: Cronograma de Atividades

Atividades	2008					2009						
	Ago	Set	Out	Nov	Dez	Jan	Fev	Mar	Abr	Mai	Jun	Jul
Revisão Bibliográfica	X											
Desenvolvimento		X	X	X	X	X	X					
Experimentação						X	X	X	X	X		
Avaliação dos Resultados									X	X	X	X

Revisão Bibliográfica

Nesta seção estão relacionados os conceitos estudados sobre análise de impacto de mudanças, bem como aplicação de probabilidade e estatística no estudo de caso.

Análise de Impacto de Mudanças

A Análise de Impacto de Mudanças estima os efeitos de uma mudança num dado sistema, produzindo resultados que auxiliam na redução dos riscos e dos custos associados à estimativa errônea desses efeitos (HUANG & SONG, 2007). Esta análise pode ser feita antes ou depois da implementação da mudança.

Quando utilizada anterior à implementação, a análise auxilia no planejamento da mudança, uma vez que permite aos desenvolvedores determinar os custos da mudança, além de ter a possibilidade de escolher soluções que causam menor impacto. Em contrapartida, quando utilizada posterior à implementação da mudança, a análise ajuda em processos de testes de regressão, ao identificar as possíveis entidades a serem novamente testadas (ARNOLD & BOHNER, 1996; HUANG & SONG, 2007).

As técnicas de análise de impacto se dividem em duas categorias: análise de impacto estática e dinâmica. A análise estática avalia o código-fonte do sistema a fim de identificar os impactos através da análise estrutural dos artefatos. Por outro lado, a análise dinâmica avalia os dados de execução do sistema, identificando relações entre os artefatos que ocorrem durante a execução desse sistema.

Ambas as abordagens requerem a especificação das mudanças e resultam o conjunto de impacto. O conjunto de impacto possui as entidades que possivelmente serão afetadas caso a mudança seja realizada. De acordo com a abordagem, o conjunto resultante pode ser denominado conjunto de impacto estático ou dinâmico.

Análise de Impacto Estática

A Análise Estática identifica relações de dependência de acordo com a estrutura dos artefatos e ocorre, geralmente, a partir da análise do código-fonte. As técnicas de análise estática normalmente mapeiam o sistema para um conjunto de grafos, no qual os nós representam as entidades e as arestas, as dependências entre estas entidades. Adicionalmente, utilizam técnicas de fechamento transitivo em grafos de chamadas ou técnicas semelhantes a fim de identificar as chamadas possivelmente impactadas por uma mudança (BREECH et al., 2005).

Um dos problemas da análise estática é que seu resultado pode atingir cerca de 90% do código do sistema, pois considera todos os comportamentos do sistema, inclusive aqueles improváveis de acontecer (ORSO et al., 2003). Dessa forma, podemos dizer que o conjunto de impacto estático pode apresentar uma grande quantidade de falso-positivos. Em contextos especiais, como em sistemas desenvolvidos em linguagens orientadas a objeto, o conjunto de impacto pode apresentar muitos falso-negativos.

Análise de Impacto Dinâmica

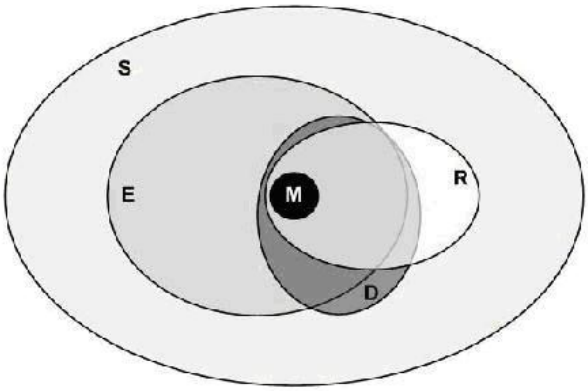
A Análise Dinâmica identifica a dependência entre as entidades através dos dados de execução do sistema, e são geralmente utilizadas após a implementação da mudança. Os dados de execução podem ser coletados para posterior análise, ou podem ser analisados durante a execução; assim, as técnicas de análise de impacto dinâmica são classificados em análise *offline* ou *online*, respectivamente (BREECH et al., 2005).

A análise dinâmica tende a obter resultados mais precisos, pois leva em consideração o real comportamento do sistema. Porém, este tipo de análise tem forte dependência com o nível de abrangência dos casos de teste; ou seja, desde que a execução tenha uma suficiente cobertura de entidades executadas, o conjunto de impacto dinâmico tende a incluir falso-negativos.

Falso-positivos e Falso-negativos

De acordo com a Figura 1, podemos considerar que o conjunto *S* é o conjunto de todas as entidades do sistema; *M*, o conjunto de mudanças e subconjunto de *S*, causa um impacto *R* nessas entidades. Os conjuntos *E* e *D* representam os resultados obtidos pelas técnicas estática e dinâmica, respectivamente. Os falso-positivos obtidos pelas análises estática e dinâmica são representadas pelos conjuntos $FPe = E - R$ e $FPd = D - R$, enquanto que os falso-negativos obtidos pelas análises são representados pelos conjuntos $FNe = R - E$ e $FNd = R - D$. A figura mostra a diferença entre os conjuntos de impacto estático e dinâmico, e que ambos incluem parte do conjunto de entidades realmente impactadas. Portanto, para realizarmos uma maior aproximação entre o conjunto de impacto real e o encontrado, é preciso unis as duas técnicas de análise.

Figura 1: Diagrama de Impacto



Fundamentação Teórica

Os conceitos abaixo citados fazem parte da técnica proposta na tese de mestrado de Mirna Carelli (MAIA, 2009), e foram implementados durante o desenvolvimento da ferramenta ImpalaDynamic. A técnica proposta realiza análise de impacto estática e pondera os resultados dessa análise de acordo com as dependências identificadas pela análise de impacto dinâmica; neste caso, esta técnica é chamada de híbrida. Os conteitos a seguir são básicos para o entendimento da ferramenta desenvolvida neste trabalho.

Evento

Consideramos um evento como a ocorrência de uma execução por um trecho do programa. Em nossa análise, que é feita em sistemas desenvolvidos em linguagens orientadas a objeto, definimos um evento como: início da execução de um método, incluindo construtor (*entry*); término da execução de um método ou construtor (*return*); início e seguinte fim da execução de um método ou construtor (*entry-return*); leitura do valor de um atributo (*read*); ou atribuição de um valor a um atributo (*write*).

Rastro de Execução

Rastro de Execução é representado por uma seqüência de eventos, responsáveis por uma determinada computação. No contexto de desenvolvimento de *software*, podemos considerar um rastro de execução como o conjunto de eventos durante a execução de um sistema. Dessa forma, sejam n eventos coletados a partir da computação de um programa e T um rastro dessa computação, então:

$$T : E_1\ E_2\ E_3\ \ldots\ E_n$$

Sucessor

Sejam A e B eventos pertencentes a um rastro de execução T de um programa, e este rastro contém n eventos. Dizemos que B é sucessor de A se e somente se:

$$T : E_1\ E_2\ E_3\ \ldots\ E_i = A\ \ldots\ Ej = B\ \ldots\ E_n,\ \text{onde } i, j < n$$

Consideramos também que B é um k-sucessor de A, ou que B é sucessor de A com distância de execução k, se ocorreram k eventos a partir de A até a primeira ocorrência de B. No exemplo acima, B é (j – (i + 1))-sucessor de A, pois o evento A não é considerado na contagem.

Distância de Sucessão

É a quantidade máxima de sucessores coletados após a execução do evento da mudança. Assim, seja um evento A; ao considerar uma distância de sucessão *n*, estamos coletando, no máximo, os *n* eventos que sucedem cada execução de A, excluindo o próprio evento A.

Mapa de Sucessão

Os sucessores dos eventos da mudança são armazenados em uma estrutura de mapa, conforme a Tabela 2. O mapa de sucessão foi implementado a partir da classe *HashMap* da biblioteca Java (HASHMAP), onde para cada entidade da mudança (chave) temos uma lista de eventos que são sucessores no determinado rastro de execução (valor). Estes sucessores são representados por pares, formados pela distância de execução e quantidade de ocorrências como sucessor nesta distância.

A Tabela 2 mostra a estrutura do mapa de sucessão na execução de um caso básico. Temos que a mudança C1 tem três eventos como sucessores: E1, E5 e E7. O evento E1 foi 0-sucessor, ou sucessor imediato da mudança nove vezes durante a execução; o evento E5 foi sucessor de distância 3 durante duas vezes, e o evento E7 por sua vez foi 4-sucessor uma única vez em toda a execução.

Tabela 2: Representação do Mapa de Sucessão

Mudança	Sucessores		
C_1	(E_1, (0, 9))	(E_5, (3, 2))	(E_7, (4, 1))
C_2	(E_2, (0, 2))	(E_1, (7, 4))	
C_3	(E_7, (1, 4))	(E_2, (5, 8))	(E_1, (6, 1))

Probabilidade de Impacto

O cálculo da probabilidade de um evento ser realmente impactado é baseado no teorema de Bayes, que calcula a probabilidade condicional de um evento a posteriori ocorrer dado um evento a priori. Para obter esse valor, assumimos os seguintes fatos:

- Cada evento identificado no rastro de execução tem a informação da quantidade de ocorrências neste rastro. Ou seja, se em um rastro *T* identificamos n eventos A, dizemos que A teve ocorrência *n*;
- Para cada sucessor de uma mudança é possível calcular um valor denominado média de ocorrência, representada pela razão entre o total de ocorrências do evento como sucessor e a quantidade de diferentes ocorrências (i.e. ocorrências com diferentes distâncias de execução) no rastro de execução.

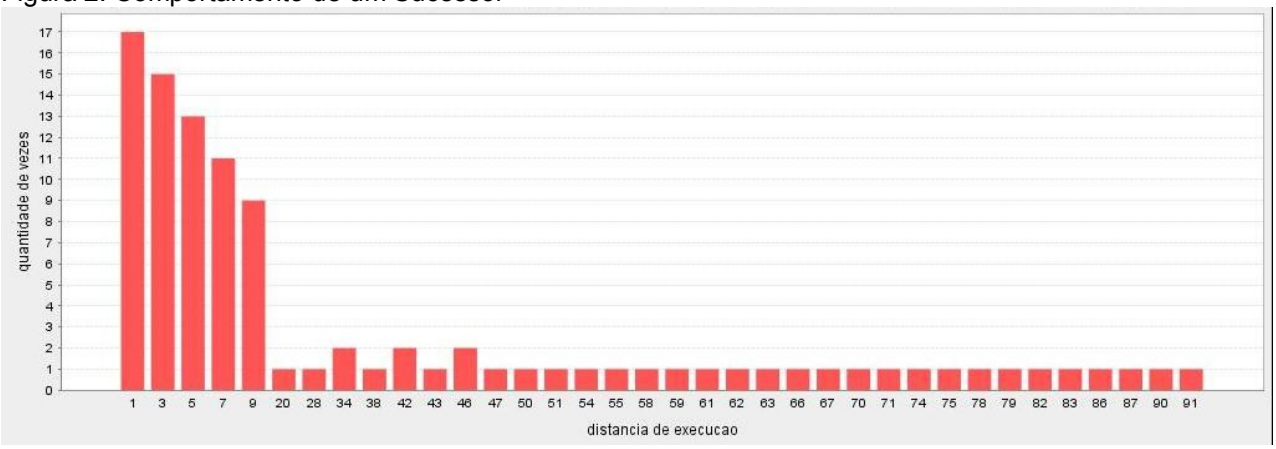
A Figura 2 representa o comportamento de um evento como sucessor em um rastro de execução de teste, onde o eixo das abcissas representa a distância de execução e o eixo das ordenadas, a quantidade de ocorrências do evento como sucessor para cada distância de execução. Portanto, a quantidade de ocorrências com diferentes distâncias de execução tem valor 36, enquanto o total de ocorrências nesse caso é:

$$t(s) = (17 + 15 + 13 + 11 + 9 + (3 \times 2) + (28 \times 1)) = 99$$

A principal importância do cálculo da média é de reduzir informações que minimizam o cálculo da probabilidade. Por exemplo, vemos na Figura 2 que o evento tem uma ocorrência muito maior com distâncias de sucessão entre 1 e 9, enquanto com outras distâncias a ocorrência é menor e, portanto, não são consideradas no cálculo. Assim, uma vez calculada a média de ocorrência, o cálculo da probabilidade é descrito a seguir:

$$P(e) = \frac{P(sucessor)}{P(sucessor) + P(não\_sucessor)}$$, onde:

Figura 2: Comportamento de um Sucessor



- *P(e)* ou probabilidade de impacto é a probabilidade que um evento tem de ser impactado, caso a mudança ocorra;
- *P(sucessor)* ou probabilidade de um evento ser sucessor é representada como a razão entre a quantidade de ocorrências como sucessor (acima da média de sucessão) e a quantidade total de ocorrências no rastro de execução;
- *P(não sucessor)* ou probabilidade de um evento não ser sucessor é por sua vez representada como a razão entre a quantidade de vezes que o evento não foi sucessor da mudança e a quantidade total de ocorrências no rastro de execução.

Desenvolvimento da Ferramenta

A ferramenta desenvolvida nesse trabalho se baseia na técnica proposta na tese de mestrado de Mirna Carelli (MAIA, 2009), integrante do Grupo de Métodos Formais da Universidade Federal de Campina Grande. A técnica realiza a combinação de técnicas estática e dinâmica em um sistema orientado a objeto para identificar as entidades possivelmente impactadas por uma mudança. O ImpalaDynamic é a ferramenta, desenvolvida em linguagem Java, que utiliza análise de impacto anterior à mudança, realizando a atribuição de probabilidades de impacto aos resultados obtidos pela técnica estática.

Especificação das Mudanças

O ImpalaDynamic recebe como entrada um conjunto de mudanças propostas, seguindo uma linguagem de descrição de mudanças, como mostra a Tabela 3. As mudanças são descritas por um tipo e a entidade a ser mudada, em que o tipo pode ser: adição, remoção ou alteração; e uma entidade pode ser uma classe, um método ou um atributo. No caso especial de remoção de classe, interpretamos que todos os seus atributos e métodos também são removidos e, consequentemente, inseridos no conjunto de mudanças.

Tabela 3: Definição da Linguagem de Descrição de Mudanças

Tipo da Mudança	Notação
Adicionar Atributo	add <nomeCompletoDaClasse> <nomeDoAtributo>
Adicionar Classe	add <nomeCompletoDaClasse>
Adicionar Método	add <nomeCompletoDaClasse> <nomeDoMétodo><argumentos>
Mudar Assinatura do Método	changeSignature <nomeCompletoDaClasse> <nomeDoMétodo><argumentos> to <novoNomeDoMétodo><argumentos>
Mudar Atributo	changeField <nomeCompletoDaClasse> <nomeDoAtributo>
Mudar Corpo do Método	changeSemantic <nomeCompletoDaClasse> <nomeDoMétodo><argumentos>
Remover Atributo	remove <nomeCompletoDaClasse> <nomeDoAtributo>
Remover Classe	remove <nomeCompletoDaClasse>
Remover Método	remove <nomeCompletoDaClasse> <nomeDoMétodo><argumentos>

ImpalaDynamic

Em linhas gerais, a ferramenta ImpalaDynamic recebe como entrada o conjunto das mudanças propostas e o sistema, identifica separadamente as entidades possivelmente impactadas pela análise estática e dinâmica e, por fim, pondera o conjunto de impacto estático com as informações obtidas na análise dinâmica. O conjunto de mudanças propostas é representado por um arquivo de texto que segue uma especificação pré-definida de acordo com o tipo de mudança. O sistema a ser analisado é recebido como um arquivo no formato *.jar* e passa por um processo de extração, através da biblioteca DesignWizard (BRUNET, 2007), desenvolvida pelo Grupo de Métodos Formais da Universidade Federal de Campina Grande.

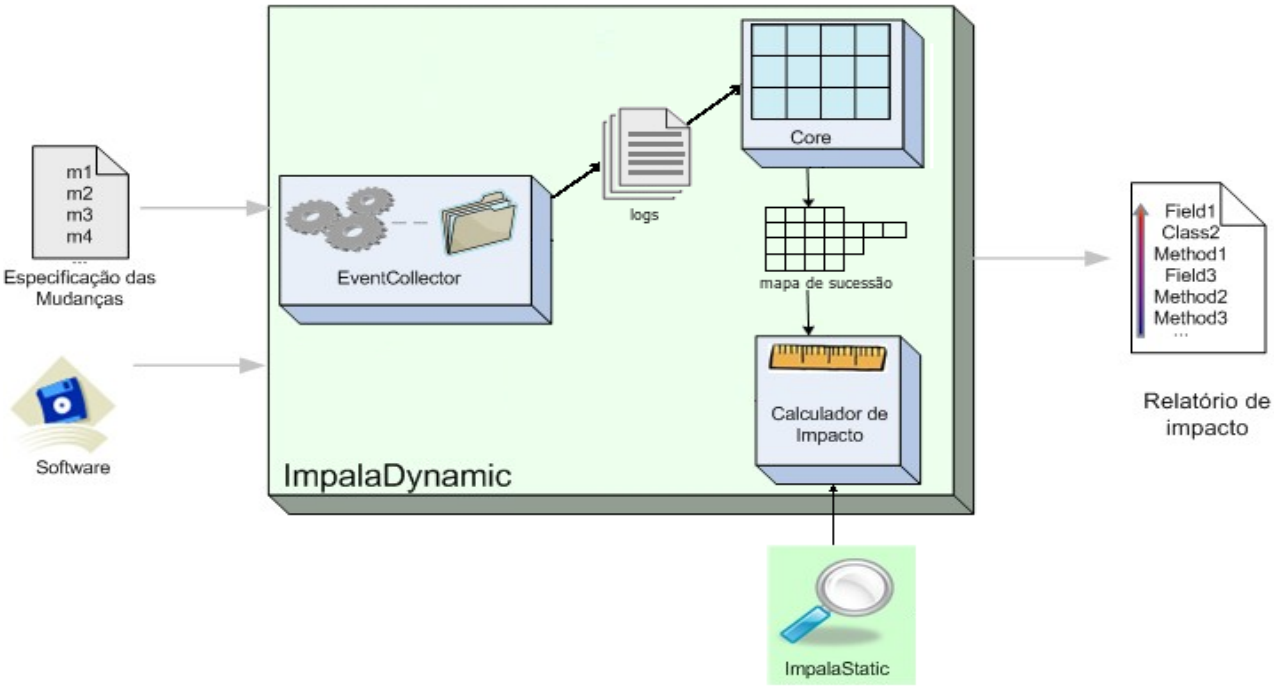
Após a etapa de extração do arquivo, o sistema passa a ser representado como um conjunto de entidades e relacionamentos. Essa etapa de extração é essencial para a posterior realização da análise estática. A análise de impacto estática é provida pela ferramenta ImpalaStatic (HATTORI, 2008). O ImpalaStatic é um analisador de impacto estático desenvolvido pelo Grupo de Métodos Formais, dentro do projeto DesignChecker, e foi escolhido pela sua fácil adaptação e acesso. A análise estática utilizada pelo ImpalaStatic resulta em um conjunto de entidades possivelmente impactadas, denominada conjunto de impacto estático.

A atribuição de probabilidade é baseada na dependência dinâmica entre as entidades, onde é realizada a análise *offline*. A partir da extração dessas dependências, é construído um mapa que corresponde a todas as execuções de entidades após a execução da mudança. Esse mapa, adicionados o conjunto de impacto estático e o conjunto de mudanças, é a entrada para o calculador de probabilidades, que pondera através de cálculos probabilísticos o conjunto de impacto estático e gera como resultado o relatório de impacto.

A Figura 3 representa uma visão geral do ImpalaDynamic, que possui três módulos principais:

- Coleta do Rastro de Execução;
- Processamento de Eventos;
- Ponderação de Impactos.

Figura 3: Visão geral da ferramenta ImpalaDynamic



Coleta do Rastro de Execução

Esse módulo é responsável por instrumentar o código do sistema para a captura de eventos durante a execução do programa. Os dados de execução são armazenados em arquivos de *log*, para serem posteriormente usados na análise de impacto dinâmica. Escolhemos o AspectJ (LADDAD, 2003; ASPECTJ) para a instrumentação do código, devido ao seu amplo uso na comunidade e a fácil configuração.

A Figura 4 ilustra o código do módulo de coleta do rastro de execução, também chamado de *Tracing*. O ponto de corte *traceMethods*, por exemplo, é responsável por capturar qualquer evento relacionado à chamada de métodos, excluindo os métodos da biblioteca Java. Enquanto que o ponto de core *traceAttributeDefinition* é responsável pela coleta de eventos relacionados à atribuição de valores a atributos, excluindo os atributos da biblioteca Java.

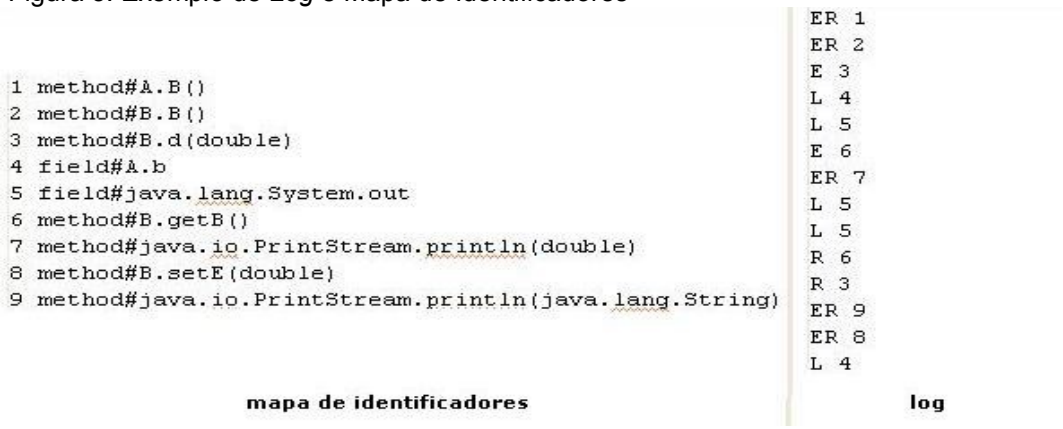
Figura 4: Captura de Eventos

```
pointcut traceMethods() : (within(project..*) && !within(java..*) &&
        !within(Tracing) && !call(private * java..* (..)) && call(* project..*(..)));

pointcut traceConstructor() : (within(project..*) && !within(java..*) &&
        !within(Tracing) && !initialization(java..new(..)) && initialization(project..new(..)));

pointcut traceAttributeAccess() : (within(project..*) && !within(java..*) &&
        !within(Tracing) && !get(* java..*) && get(* project..*));

pointcut traceAttributeDefinition() : (within(project..*) &&
        !within(java..*) && !within(Tracing) && !set(* java..*) && set(* project..*) );
```

Após a coleção de eventos são gerados dois artefatos: o *log*, contendo todos os eventos na ordem em que eles ocorreram na execução, e o mapa de identificadores, onde para cada identificador (chave) é atribuída uma única entidade (valor). No *log* também são armazenados, para cada evento, um identificador

indicando qual o tipo de evento que ocorreu, seguindo a definição de Evento na Fundamentação Teórica. Estes artefatos são mostrados na Figura 5.

Figura 5: Exemplo de *Log* e Mapa de Identificadores



No exemplo acima, vemos que a entidade de identificador 2 (o método A.B( )) foi o segundo evento a ocorrer durante a execução do programa teste. Vemos também que esta ocorrência foi uma entrada e seguinte retorno do método A.B( ), indicado no *log* pelo identificador "ER".

Processamento de Eventos

O módulo de processamento de eventos do ImpalaDynamic, também chamado de *Core*, é responsável por armazenar os sucessores dos eventos relacionados às entidades mudadas. Este módulo recebe como entrada o conjunto de mudanças e os dados de execução do programa, e o processamento é descrito a seguir:

- Seja *T* um rastro de execução, *M* o conjunto de mudanças e *S*, o mapa de sucessão;
- Seja *c* o último evento de mudança identificado no rastro;
- A coleta de sucessores não foi iniciada e a distância de execução *d* tem valor inicial 0;
- Para cada evento *e* no rastro de execução:
  - Se $e \in M$:
    - *c* recebe *e*;
    - inicia a coleta de sucessores e *d* é igual a 0.
  - Caso contrário, se a coleta de sucessores foi iniciada:
    - adiciona *e* a *S* como sucessor de *c*, com distância de execução *d*;
    - incrementa *d*.

Ponderação de Impactos

Este módulo é responsável por ponderar o conjunto de impacto estático e gerar o relatório de impacto. O ponderador de impactos recebe como entrada o conjunto de impacto estático, gerado pela ferramenta ImpalaStatic, e o mapa de sucessão, provido pelo módulo de processamento de eventos, e realiza a ponderação da seguinte forma:

- Sejam *I* o conjunto de impacto estático e *S* o conjunto de sucessores identificados;
- Para cada $e \in I$, calcula a probabilidade de impacto *P* (vide seção Probabilidade de Impacto da Fundamentação Teórica), e insere no conjunto ponderado;
- Para cada $e \in S$ e $e \notin I$, marca como "não identificado", calcula sua probabilidade *P* de impacto e insere no conjunto ponderado.

ATIVIDADES REALIZADAS

Após o desenvolvimento do protótipo da ferramenta ImpalaDynamic, realizamos experimentação de validação da técnica. Os experimentos foram realizados com projetos desenvolvidos no Grupo de Métodos Formais da Universidade Federal de Campina Grande, e em linguagem orientada a objeto Java. A comunicação com os desenvolvedores foi fundamental para a avaliação dos resultados obtidos na análise, uma vez que o conhecimento da estrutura e do funcionamento do sistema é imprescindível para identificar os impactos reais de uma mudança. Os projetos são citados a seguir:

- Toy Example: projeto de teste desenvolvido pela mesma equipe do ImpalaDynamic. Este programa realiza adição e remoção de itens em documentos, e foi construído para verificar a eficácia da técnica proposta em sistemas com *late binding* (LATE BINDING);
- Design Suíte: realiza recuperação e verificação arquitetural de *software*. O trabalho é descrito em (BITTENCOURT et al., 2009) e é dividido em duas partes:
  - Design: responsável por oferecer uma estrutura de grafo que irá representar o sistema a ser verificado, incluindo persistência em arquivo e funções de qualidade de *design*;
  - Abstractor: responsável por realizar a combinação de entidades de baixo nível arquitetural para entidades de níveis superiores em modelos arquiteturais.
- ImpalaStatic: realiza análise de impacto estática em sistemas desenvolvidos em linguagem Java;
- DesignWizard: biblioteca de inspeção automática em programas Java.

A Tabela 4 descreve a estrutura dos projetos coletados, de acordo com a quantidade de classes, métodos e atributos

Tabela 4: Descrição estrutural dos projetos coletados

	Quantidade de Entidades			
	Classes	**Métodos**	**Atributos**	**Total**
Toy Example	19	95	20	134
Design	11	184	72	267
Abstractor	82	624	360	1066
ImpalaStatic	36	376	117	529
DesignWizard	50	708	189	947

Em reunião com os desenvolvedores de cada projeto, foram definidos o conjunto de mudanças e o impacto real dessas mudanças em cada sistema, onde foram gerados dois artefatos: um arquivo de descrição de mudanças e um arquivo com a descrição das entidades que são realmente impactadas. As mudanças seguem a linguagem de descrição de mudanças citada na seção Especificação das Mudanças. A Tabela 5 descreve as mudanças propostas em cada projeto coletado:

Tabela 5: Descrição das Mudanças Propostas

Projeto	**Mudanças**
Toy Example	changeSignature example.command.CommandWithLabel getLabel() to example.command.CommandWithLabel getLabel()
	remove example.app.Application addMenuItem(example.command.CommandWithLabel)
	changeSignature example.app.Application addMenuItem(java.lang.String) to example.app.Application addMenuItem(int)
Design	remove design.model.Design getGraph()
	changeSemantics design.model.Design getGraph(int)
	changeSemantics design.model.Design size()
	remove design.model.Design$Pair
Abstractor	remove abstractor.cluster.mq.optimization.MQTurboRanker inter
	changeSemantics abstractor.cluster.mq.optimization.MQTurboRanker getAllClusterFactor(abstractor.cluster.jung.ClusterSet)
	remove abstractor.cluster.dsm.DSMClusterer powDep
ImpalaStatic	add class org.impala.impalastatic.exceptions.EntityNotSupportedException
	add org.impala.impalastatic.exceptions.EntityNotSupportedException serialVersionUID
	changeSignature org.impala.impalastatic.algorithms.command.Command

	changeSignature org.impala.impalastatic.algorithms.command.Command execute (org.impala.impalastatic.data.ImpactedTree, org.designwizard.main.DesignWizard, int, java.lang.String[]) to void execute(org.impala.impalastatic.data.ImpactedTree, org.designwizard.main.DesignWizard, int, java.lang.String[])
	changeSemantics org.impala.impalastatic.algorithms.command.ChangeFieldCommand execute (org.impala.impalastatic.data.ImpactedTree, org.designwizard.main.DesignWizard, int, java.lang.String[])
	changeSemantics org.impala.impalastatic.algorithms.command.ChangeSemanticsCommand execute(org.impala.impalastatic.data.ImpactedTree, org.designwizard.main.DesignWizard, int, java.lang.String[])
DesignWizard	remove org.designwizard.design.MethodNode.getPackageName()

Em seguida, todos os projetos foram instrumentados e seus rastros de execução foram coletados em arquivos de *log*. A Tabela 6 descreve os rastros de execução coletados em cada projeto, de acordo com a quantidade de eventos capturados e o tamanho do arquivo de *log*.

Tabela 6: Descrição dos *Logs*

	Quantidade de Eventos	Tamanho do Log
Toy Example	640	3.2 KB
Design	103 793	823.5 KB
Abstractor	16 015 121	101.1 MB
ImpalaStatic	1 537 684	11.6 MB
DesignWizard	381 639 833	2.2 GB

Após ser definido o impacto real, as mudanças propostas foram implementadas em cada projeto. Eventuais erros de código e desenho foram corrigidos e, no final desse processo, foram disponibilizadas duas versões de cada projeto: uma antes e outra depois da implementação da mudança. A importância dessa fase é identificar o real impacto no projeto, adicionando eventuais entidades impactadas não identificadas pelos desenvolvedores.

Em seguida, realizamos a análise de impacto dinâmica com o algoritmo CollectEA, um dos mais relevantes da área. O CollectEA é o algoritmo de análise dinâmica mais eficiente (ORSO et al., 2004) e baseia-se na relação binária, chamada de *execute after*, entre duas entidades *a* e *b* de um programa P: se *a* é executado após *b* em alguma execução de P, então $(a, b) \in EA$ (HUANG & SONG, 2006; APPIWATTANAPONG et al., 2005). O algoritmo armazena o tempo de execução de cada evento, os ordena e identifica as relações *execute after* entre as entidades. O conjunto de impacto é formado pelas entidades que foram executadas após a mudança. A importância do uso desse algoritmo é de gerar um comparativo dos resultados entre o conjunto de impacto obtido e o conjunto gerado pelo módulo processador de eventos do ImpalaDynamic.

Para a análise estática com a ferramenta ImpalaStatic, utilizamos a representação do sistema em grafo, extraído pela ferramenta DesignWizard, e um parâmetro de profundidade. Este parâmetro define o quão a busca no grafo por dependência será profunda, isto é, quantos níveis do grafo serão analisados. Para nosso experimento, trabalhamos com a seguinte variação de parâmetros: 5, 10 e 15 níveis. É importante ressaltar que a profundidade está diretamente relacionada com a abrangência da análise estática, pois quanto maior a profundidade, maior será o conjunto de entidades possivelmente impactadas.

Por fim, o ImpalaDynamic foi executado em os dados coletados, e o relatório de impacto (com o conjunto ponderado) foi combinado com o conjunto de impacto real, resultando no relatório de experimento. Para o nosso experimento, executamos o módulo de processamento de eventos do ImpalaDynamic com as seguintes distâncias de sucessão (descrita na Fundamentação Teórica): 100, 500 e 1000 sucessores. O relatório de experimento contém a proporção e a quantidade total de acertos das técnicas estática e dinâmica.

RESULTADOS E DISCUSSÃO

Os resultados obtidos nos relatórios de experimento foram disponibilizados em tabelas, mostradas a seguir. A primeira coluna denota a técnica de análise de impacto utilizada, incluindo a técnica construída nesse trabalho, chamada “Híbrida”. A coluna “Acertos” mostra a quantidade de entidades realmente impactadas identificadas em cada técnica, de acordo com o impacto real definido no início da

experimentação. Além disso, a coluna “Acertos/R” mostra a proporção entre os acertos e o impacto real. Este dado é importante, pois quanto maior a proporção de acertos, menor a quantidade de falso-negativos.

A Tabela 7 descreve os resultados da análise dinâmica realizada pelo algoritmo CollectEA em cada projeto:

Tabela 7: Resultado do algoritmo CollectEA

CollectEA				
Projeto	**Acertos**	**Conjunto de Impacto**	**Acertos/R (%)**	**Impacto Real**
ToyExample	19	84	59,37	32
Design	5	46	13,88	36
Abstractor	13	215	72.22	18
ImpalaStatic	6	800	9,23	65
DesignWizard	4	289	57,14	7

As tabelas seguintes descrevem os resultados da análise estática em cada projeto. A análise estática foi realizada pela ferramenta ImpalaStatic e com variação de parâmetro de profundidade igual a [5, 10, 15] níveis:

Tabela 8: Resultado da Análise Estática para o projeto Toy Example

Análise Estática – Toy Example				
Profundidade	**Acertos**	**Conjunto de Impacto**	**Acertos/R (%)**	**Impacto Real**
5	5	26	15,62	32
10	5	26	15,62	32
15	5	26	15,62	32

Tabela 9: Resultado da Análise Estática para o projeto Design

Análise Estática – Design				
Profundidade	**Acertos**	**Conjunto de Impacto**	**Acertos/R**	**Impacto Real**
5	11	20	30,55	36
10	11	20	30,55	36
15	11	20	30,55	36

Tabela 10: Resultado da Análise Estática para o projeto Abstractor

Análise Estática – Abstractor				
Profundidade	**Acertos**	**Conjunto de Impacto**	**Acertos/R**	**Impacto Real**
5	2	13	11,11	18
10	2	17	11,11	18
15	2	17	11,11	18

Tabela 11: Resultado da Análise Estática para o projeto ImpalaStatic

Análise Estática – ImpalaStatic				
Profundidade	**Acertos**	**Conjunto de Impacto**	**Acertos/R**	**Impacto Real**
5	41	70	63,07	65
10	42	83	64,61	65
15	43	84	66,15	65

Tabela 12: Resultado da Análise Estática para o projeto DesignWizard

Análise Estática – DesignWizard				
Profundidade	**Acertos**	**Conjunto de Impacto**	**Acertos/R**	**Impacto Real**
5	1	2	14,28	7
10	1	2	14,28	7
15	1	2	14,28	7

Por fim, as tabelas a seguir descrevem os resultados da análise realizada pelo ImpalaDynamic, de acordo com as diferentes distâncias de sucessão utilizadas nos experimentos. Uma vez que o ImpalaDynamic faz uso da análise estática da ferramenta ImpalaStatic, adicionamos um segundo parâmetro no ImpalaDynamic, referente à profundidade da análise estática.

Tabela 13: Resultado da análise híbrida para o projeto Toy Example

ImpalaDynamic – Toy Example					
Profundidade	**Dist. de Sucessão**	**Acertos**	**Conj. de Impacto**	**Acertos/R**	**Impacto Real**
5	100	12	73	37,5	32
	500	12	73	37,5	32
	1000	12	73	37,5	32
10	100	18	95	56,25	32
	500	18	95	56,25	32
	1000	18	95	56,25	32
15	100	18	95	56,25	32
	500	18	95	56,25	32
	1000	18	95	56,25	32

Tabela 14: Resultado da análise híbrida para o projeto Design

ImpalaDynamic – Design					
Profundidade	**Dist. de Sucessão**	**Acertos**	**Conj. de Impacto**	**Acertos/R**	**Impacto Real**
5	100	14	49	38,88	36
	500	14	52	38,88	36
	1000	14	52	38,88	36
10	100	14	49	38,88	36
	500	14	52	38,88	36
	1000	14	52	38,88	36
15	100	14	49	38,88	36
	500	14	52	38,88	36
	1000	14	52	38,88	36

Tabela 15: Resultado da análise híbrida para o projeto Abstractor

ImpalaDynamic – Abstractor					
Profundidade	**Dist. de Sucessão**	**Acertos**	**Conj. de Impacto**	**Acertos/R**	**Impacto Real**
5	100	10	147	55,55	18
	500	10	166	55,55	18
	1000	10	182	55,55	18
10	100	10	151	55,55	18
	500	10	170	55,55	18
	1000	10	186	55,55	18
15	100	10	151	55,55	18
	500	10	170	55,55	18
	1000	10	186	55,55	18

Tabela 16: Resultado da análise híbrida para o projeto ImpalaStatic

ImpalaDynamic – ImpalaStatic					
Profundidade	**Dist. de Sucessão**	**Acertos**	**Conj. de Impacto**	**Acertos/R**	**Impacto Real**
5	100	44	247	67,69	65
	500	45	291	69,23	65
	1000	45	353	69,23	65
10	100	42	150	64,61	65
	500	46	198	70,76	65
	1000	46	237	70,76	65
15	100	46	261	70,76	65
	500	47	304	72,30	65
	1000	47	366	72,30	65

Tabela 17: Resultado da análise híbrida para o projeto DesignWizard

ImpalaDynamic – DesignWizard					
Profundidade	**Dist. de Sucessão**	**Acertos**	**Conj. de Impacto**	**Acertos/R**	**Impacto Real**
5	100	4	65	57,14	7
	500	5	132	71,42	7
	1000	5	160	71,42	7
10	100	4	65	57,14	7
	500	5	132	71,42	7
	1000	5	160	71,42	7
15	100	4	65	57,14	7
	500	5	132	71,42	7
	1000	5	160	71,42	7

Observamos que, nos experimentos relacionados aos projetos Design, ImpalaStatic e DesignWizard, houve uma significante diferença entre os resultados do ImpalaDynamic e as técnicas estática e dinâmica separadamente. Em dois desses casos, nos projetos ImpalaStatic e DesignWizard, observamos que os resultados identificaram mais da metade do impacto real.

Em outros casos, nos projetos ToyExample e Abstractor, notamos uma melhor proporção de acertos com relação à técnica dinâmica. É notável que o conjunto de impacto híbrido cresce proporcionalmente à distância de sucessão e, consequentemente, reduzindo a quantidade de falso-negativos. Espera-se que quanto maior a distância de sucessão, mais próximo será o resultado com a análise dinâmica com CollectEA, uma vez que este último inclui todos os eventos posteriores ao evento da mudança como possível impacto.

Durante a execução da análise dinâmica, identificamos que a execução do programa analisado pode não ser abrangente o suficiente para incluir a execução das entidades realmente impactadas. A Tabela 18 descreve a cobertura de código dos projetos coletados, isto é, qual porcentagem do código foi executada na coleta do rastro de execução.

Tabela 18: Cobertura de Código dos projetos coletados

Cobertura de Código	
Projeto	**Cobertura (em %)**
Toy Example	84,22
Design	52,49
Abstractor	64,80
ImpalaStatic	63,63
DesignWizard	71,10

Assim, ao analisar a cobertura de código, as entidades que não foram executadas não serão incluídas no conjunto de impacto, e provavelmente reduziremos a quantidade de falso-negativos à medida que a cobertura de código for maior.

PIBIC/CNPq/UFCG-2009

CONCLUSÕES

Foi possível concluir que o trabalho realizado foi produtivo, tanto em relação ao conhecimento adquirido na área de Análise de Impacto quanto aos artefatos produzidos, a exemplo da ferramenta desenvolvida e os resultados obtidos com os experimentos. A possibilidade de efetuar análise de impacto de mudanças é de grande importância para os desenvolvedores e arquitetos na estimativa do impacto em sistemas de *software* reais.

Uma vez que permitimos que esta análise seja feita antes da implementação da mudança, estamos auxiliando no planejamento da mesma em contextos de evolução de *software* e planejamento de recursos de mão-de-obra em processos evolutivos.

Ao combinar as técnicas estática e dinâmica, espera-se que o conjunto de impacto obtido reduza a quantidade de falso-negativos, permitindo assim que o conjunto seja mais realista. Esta redução possui grande importância pois, uma vez que ignoramos falso-negativos, podemos estar ignorando possíveis impactos reais.

A partir dos resultados dos experimentos, podemos concluir que a distância de sucessão é um parâmetro fundamental na ferramenta ImpalaDynamic, pois permite, na maioria dos casos, que sejam identificadas mais entidades realmente impactadas, o que implica em redução dos falso-negativos. Adicionalmente, consideramos que a abrangência da execução, ou cobertura de código, do programa analisado é de grande relevância nos resultados e também na redução dos falso-negativos, pois assim estaríamos analisando os outros possíveis comportamentos do sistema, incluindo aqueles em que a mudança teria maior impacto.

AGRADECIMENTOS

Ao CNPq pela bolsa de Iniciação Científica e a Dalton Serey, Mirna Carelli e Isabel Nunes pela dedicação e orientação empregados para que o projeto fosse realizado.

REFERÊNCIAS BIBLIOGRÁFICAS

APIWATTANAPONG et al., 2005. Apiwattanapong, T.; Orso, A.; Harrold, M. J. **Efficient and precise dynamic impact analysis using execute-after sequences**. In: ICSE '05: Proceedings of the 27th international Conference on Software engineering. New York, NY, USA: ACM, 2005. p. 432–441. ISBN 1-59593-963-2.

ARNOLD & BOHNER 1996. Robert S. Arnold and Shawn Bohner. **Software Change Impact Analysis**. IEEE. Computer Society Press, Los Alamitos, CA, USA, 1996.

BITTENCOURT et al., 2009. Bittencourt, R. A. ; Damásio, J. F. ; Santos, G. J. S. ; Almeida Filho, A. T. ; Nóbrega Filho, J. M. ; Figueiredo, J. C. A. ; Guerrero, D. D. S. . **Design Suite: Towards an Open Scientific Investigation Environment for Software Architecture Recovery**, 2009, Ouro Preto. Anais do VIII Simpósio Brasileiro de Qualidade de Software. Rio de Janeiro : UFF, 2009.

BOHNER, 2002. Bohner, Shawn A. **Software change impacts - an evolving perspective**. In ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02), pages 263–2, Washington, DC, USA, 2002. IEEE Computer Society.

BREECH et al., 2005. Breech, B., Tegtmeyer, M., and Pollock, L. **A comparison of online and dynamic impact analysis algorithms**. In CSMR '05: Proceedings of the Ninth European conference on Software Maintenance and Reengineering, pages 143–152, Washington, DC, USA. IEEE Computer Society.

MAIA, 2009. Maia, Mirna C. O. **Uma técnica de análise de impacto de mudanças baseada na análise probabilística dos dados obtidos em tempo de execução**. Tese de Mestrado, Universidade Federal de Campina Grande, Brasil.

ERLIKH, 2000. Erlikh, L. **Leveraging legacy system dollars for e-business**. IT Professional, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 2, n. 3, p. 17–23, 2000. ISSN 1520-9202.

HATTORI, 2008. Hattori, L. P. **Análise probabilística de impacto de mudanças baseada em históricos de mudanças de software**. Tese de Mestrado, Universidade Federal de Campina Grande, Campina Grande, Brasil.

HUANG & SONG, 2006. HUANG, L.; SONG, Y.-T. **Dynamic impact analysis using execution profile tracing**. sera, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 237–244, 2006.

HUANG & SONG, 2007. Huang, L. and Song, Y.-T. **Precise dynamic impact analysis with dependency analysis for object-oriented programs**. In SERA ’07: Proceedings of the 5th CIS International Conference on Software Engineering Research, Management & Applications (SERA 2007), pages 374–384, Washington, DC, USA. IEEE Computer Society.

LADDAD, 2003. Laddad, R. (2003). **Aspectj in action: Practical Aspect-Oriented Programming**. Manning Publications Co.

LEHMAN, 1980. Lehman, Meir. **Programs, life cycles and the laws of software evolution**. In Proc. IEEE, Special Issue on Software Evolution, pages 1060–1076. IEEE, 1980

ORSO et al., 2003. Orso, A.; Apiwattanapong, T.; Harrold, M. J. **Leveraging field data for impact analysis and regression testing**. In: ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY, USA: ACM, 2003. p. 128–137. ISBN 1-58113-743-5

ORSO et al., 2004. Orso, A. et al. **An empirical comparison of dynamic impact analysis algorithms**. In: ICSE ’04: Proceedings of the 26th International Conference on Software Engineering. Washington, DC, USA: IEEE Computer Society, 2004. p. 491–500. ISBN 0-7695-2163-0.

ASPECTJ. **AspectJ**. Disponível em: <http://www.eclipse.org/aspectj/>. Acesso em 27 de julho de 2009.

DESIGNWIZARD, 2008. **DesignWizard**. Disponível em <http://www.designwizard.org/>. Acesso em 27 de julho de 2009.

HASHMAP, 2008. **API da classe java.util.HashMap**. Disponível em <http://java.sun.com/j2se/1.3/docs/api/java/util/HashMap.html>. Acesso em 27 de julho de 2009.

LATE BINDING. **Java – Sobrecarga de Construtores**J. Disponível em <http://www.inf.puc-rio.br/~java/progjava/protected/apostilas/oo2.pdf>. Acesso em 27 de julho de 2009.