Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Informática

Técnica Híbrida de Análise de Impacto para Sistemas Orientados a Objetos

Mirna Carelli Oliveira Maia

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Engenharia de Software

Dalton Dario Serey Guerrero
(Orientador)
Jorge César Abrantes de Figueiredo
(Orientador)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

M217t
2009 Maia, Mirna Carelli Oliveira.
Técnica híbrica de análise de impacto para sistemas orientados a objetos /Mirna Carelli Oliveira Maia. ─ Campina Grande, 2009.
75 f.: il.

Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Referências.
Orientadores: Prof. Dr. Dalton Dario Serey e Prof. Dr. Jorge César Abrantes de Figueiredo.

1. Manutenção de Programas. 2. Análise de Impacto. 3. Impacto de Mudanças. I. Título.

CDU 621.39(043)

"TÉCNICA HÍBRICA DE ANÁLISE DE IMPACTO PARA SISTEMAS ORIENTADOS A OBJETOS"

MIRNA CARELLI OLIVEIRA MAIA

DISSERTAÇÃO APROVADA EM 17.08.2009

DALTON DARIO SEREY GUERRERO, D.SC
Orientador(a)

JORGE CESAR ABRANTES DE FIGUEIREDO, D.SC
Orientador(a)

ROHIT GHEYI, DR.
Examinador(a)

MARCOS LORDELLO CHAIM , DR.
Examinador(a)

CAMPINA GRANDE - PB

Resumo

Durante o processo de desenvolvimento de software, mudanças ocorrem para que os requisitos continuem atualizados de acordo com as necessidades dos clientes. A implementação de mudanças é uma atividade cara e não trivial. A análise de impacto é o processo de identificação das consequências da mudança no programa. A análise de impacto pode utilizar técnicas que utilizam a análise estática ou dinâmica para identificação do impacto das mudanças. Essas que podem produzir resultados imprecisos porque superestimam ou subestimam o impacto. No primeiro caso, podem incluir entidades não-afetadas caracterizando a presença de falso-positivos, o que pode fazer a empresa perder negócios para concorrentes. No segundo caso, os resultados desconsideram entidades afetadas, o que caracteriza a presença de falso-negativos, o que pode representar que a empresa vai ter prejuízos financeiros. Na maioria dos contextos, o prejuízo associado aos falso-negativos é maior do que os falso-positivos. Neste trabalho, propomos e avaliamos uma técnica de análise de impacto que visa reduzir o número de falso-negativos. Avaliamos a técnica segundo dois critérios: viabilidade e efetividade. Para avaliar a viabilidade, implementamos o protótipo SD-Impala para representar as etapas da técnica proposta. Para avaliar a efetividade, realizamos a análise de impacto em projetos reais utilizando ferramentas diferentes e comparamos os erros de seus resultados. A análise desses valores mostrou que a técnica proposta reduziu os falso-negativos entre 90% e 115% em relação à técnica estática, e reduziu entre 21,2% e 39% em relação à técnica dinâmica.

Abstract

Changes happen during the software development process so that requirements keep up-to-date with costumers needs. The change implementation is an expensive and not trivial activity. Impact analysis is the process that aids software engineers in identifying the consequences of program changes. It may be used before doing changes to calculate its costs or after changes to validate the modified program with regression tests. Two approaches are usually taken in impact analysis: static or dynamic techniques. In the former, program structure is analyzed in order to identify change impact. The latter analyzes execution traces to identify change impact based on dynamic dependencies. Both approaches may be inaccurate, super-estimating or sub-estimating the impact. In the former case, non-affected entities may be included, characterizing the presence of false-positives. In the latter, results do not take into account affected entities, characterizing false-negatives. With false-positives, unnecessary super-estimated information may confuse the analyst. On the other hand, false-negatives mean sub-estimated impact that causes financial losses to the company, usually worse than false-positives. In this work, we propose and evaluate a hybrid impact analysis technique that aims to reduce the number of false-negatives. The technique is hybrid because it combines static and dynamic approaches to identify change impact. We evaluated the technique comparing the absolute number of false negatives and the obtained recall. Recall is a metric that represents the ratio between the number of entities correctly obtained by the analysis and the total number of affected entities. Analysis of the results showed that the proposed technique increased recall between 90 and 115% compared to the static technique, and between 21,2 and 39% compared to the dynamic technique. Although our results are encouraging, a more thorough study is needed to evaluate up to where the results may be generalized.

Agradecimentos

A Deus por ser permitir que eu alcance mais essa vitória em minha vida.

A mainha por ter abdicado um pouco de viver sua vida para construir o meu cárater e minha personalidade.

A painho por ser a pessoa que mais se orgulha por essa conquista.

A Mikaela por ser minha fã número 1 (não assumida).

A meus avós, tios e primos pelo apoio e torcida. Em especial, pelo carinho dos meus queridos Tio Filhinho, Tio Clóvis e Tia Lúcia.

Às minhas amigas que estão sempre torcendo e vibrando por minhas vitórias e que têm um espaço especial no meu coração: Flávia, Flora, Ismenya, Laisa e Vyrna. Em especial, a Chá e Yuska por todos os conselhos e cuidados e, principalmente, pela paciência que tiveram comigo.

A Marcus por torcer em cada vitória e acreditar no meu triunfo (na maioria das vezes) mais do que eu mesma.

Aos companheiros que compartilham os momentos de *deadline* e aperreios: Guiga, João, Lauro, Lile, Manu e Saulo. Em especial, agradeço a Bel, Gustavo, Roberto e Zé Filho pelo incentivo e apoio durante essa jornada.

À galera que colaborou com o entretenimento (farras, viagens e gargalhadas) que foram essenciais para que eu conseguisse chegar até o final: Danilo, Hugo, Larissa Lucena, Larissa Prazeres e Renata. Em especial, a Nigini por ter sido uma peça fundamental na melhor viagem da minha vida.

Aos meus orientadores Dalton e Jorge, por guiarem nessa difícil caminhada.

Aos professores e funcionários da COPIN e do DSC.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Códigos Fonte

Capítulo 1

Introdução

Durante o processo de desenvolvimento de software os requisitos inicialmente levantados podem sofrer alterações a fim de permanecerem de acordo com as necessidades dos clientes e usuários. Essas mudanças podem ser a inserção ou alteração de requisitos, correção de erros, refatoramento de código, etc. As atividades relacionadas à implementação de mudanças são caras e consomem mais de 50% do custo total do projeto [6]. Em alguns contextos, como em sistemas orientados a objetos, esse custo pode aumentar ainda mais devido à presença de características especiais que afetam a forma de propagação dos efeitos das mudanças pelo sistema [14].

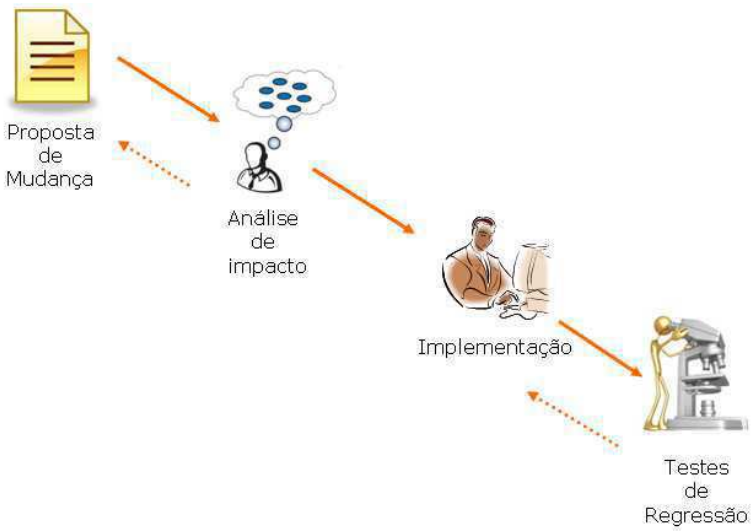


Figura 1.1: Processo de Análise de Impacto

A análise de impacto de mudanças é o processo que permite engenheiros de *software* identificar as consequências da alteração no programa [22]. A Figura 1.1 representa um processo típico de análise de impacto em uma fábrica de software, por exemplo. Nesse processo, as propostas de mudanças são passadas para o analista que é responsável por identificar o impacto porque conhece a arquitetura do sistema e pode identificar as alterações necessárias para realizar a mudança proposta. Após a identificação do impacto, ocorre a estimativa de esforço e a realização da mudança. Durante a implementação das mudanças, é possível que novos impactos sejam encontrados, representando que a estimativa não estava de acordo com a realidade. Para revalidar o software após a mudança, realiza-se testes de regressão para certificar que o comportamento do sistema não foi afetada.

Identificar o impacto de mudanças não é uma tarefa trivial. Geralmente, isso acontece por que a análise é manual, os analistas utilizam ferramentas inadequadas ou por que a quantidade de informação analisada é muito grande. Por esses motivos, o resultado da análise de impacto pode apresentar erros porque superestima o impacto ou porque subestima o impacto. No contexto de uma fábrica de software como a parceira do nosso trabalho, o prejuízo causado pela subestimativa de impacto é maior do que o prejuízo causado pela superestimativa.

Suponha uma mudança M em um programa P qualquer. Essa mudança afeta outras entidades de P que compõem o subconjunto A. O resultado da análise de impacto contém entidades consideradas afetadas e é conhecido como conjunto de impacto I. A Figura 1.2 é o diagrama de Venn que representa a relação entre o conjunto I e o conjunto das entidades efetivamente afetadas A pela mudança M no programa P. Se $I = A$ então dizemos que a análise produziu o resultado ótimo e, contém exclusivamente entidades impactadas. A obtenção da solução ótima não é uma tarefa trivial e ainda não foi alcançada devido à presença de erros nos resultados da análise de impacto [18; 9; 10]. Quando o resultado apresenta entidades não impactadas, consideramos que este possui falso-positivos. Por outro lado, quando não possui entidades impactadas, consideramos que o resultado possui falso-negativos.

Para mensurar o erro da análise podemos utilizar o número de falso-negativos e falsopositivos. Entretanto, como são valores absolutos, não podemos utilizá-los para comparar resultados em contextos diferentes. Por esse motivo, também utilizamos as métricas precisão e revocação redefinidas para o contexto de análise de impacto [7]. Precisão é a razão entre

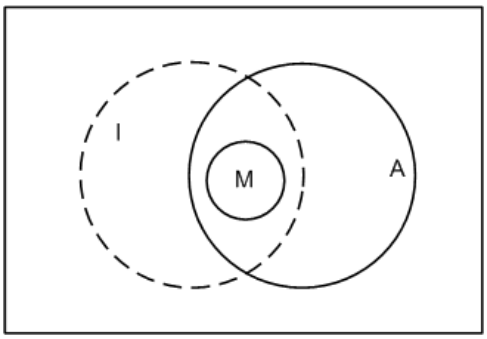


Figura 1.2: Relação entre o conjunto de impacto e as entidades afetadas pela mudança

o número de entidades obtidas corretamente pela técnica e o número total de entidades do sistema, seu valor é inversamente proporcional ao número de falso-positivos. Já revocação é a razão entre o número de entidades obtidas corretamente pela análise e o número total de entidades afetadas, seu valor é inversamente proporcional ao número de falso-negativos.

As técnicas de análise de impacto se enquadram em duas abordagens: estática ou dinâmica. Na primeira, artefatos são analisados a fim de identificar dependências estruturais entre as demais entidades do sistema e as entidades da mudança. Na segunda, a dependência é obtida pela análise de dados da execução do programa. Contudo, essas abordagens podem ser combinadas formando uma abordagem híbrida, onde o impacto é estimado pela análise da estrutura e dos dados dinâmicos do sistema.

Técnicas baseadas na análise estática tendem a superestimar o impacto incluindo entidades que não são efetivamente impactadas, os chamados falso-positivos. Isto acontece porque desconsidera o comportamento real do sistema [2; 9; 19; 10]. Em situações especiais, com presença de técnicas como very late binding onde informações são conhecidas apenas em tempo de execução, os resultados da análise estática também pode apresentar falso-negativos.

Estudos apontam a análise dinâmica como mais precisa que a estática porque analisa os rastros da execução do programa e, consequentemente, levam em consideração o comportamento do sistema [2; 9; 19; 10]. Nesse contexto, o comportamento do sistema é determinado pela análise dos rastros de execução obtidos, na maioria das vezes, através da execução de testes. Assim, o resultado da análise e sua precisão dependem diretamente dos dados de entrada para execução do programa. De acordo com os dados utilizados para execução do

programa, o resultado da análise dinâmica pode apresentar falso-negativos e falso-positivos.

Além da imprecisão, outro problema relacionado à análise de impacto é a organização do resultado gerado. Em situações onde o conjunto de impacto resultante é grande, a informação produzida pode confundir o engenheiro de software na realização da análise. Isso acontece porque é a relação de pertinência no conjunto de impacto é o critério que define se a entidade é considerada afetada ou não.

Esse trabalho tem como objetivo reduzir o número de falso-negativos encontrados por técnicas que utilizam as abordagens estática e dinâmica aplicadas separadamente. Assim, investigamos a seguinte hipótese:

A abordagem híbrida de análise de impacto permite classificar o impacto das mudanças e reduzir a quantidade de falso-negativos obtidos em relação às técnicas estática e dinâmica existentes.

Para isso, elaboramos e avaliamos uma técnica híbrida de análise de impacto. A nossa solução analisa a estrutura e os dados de execução do programa, identifica o impacto baseado no resultado dessas análises e o classifica em relação a dependência dinâmica entre as entidades. Para verificar a viabilidade, implementamos o protótipo *SD*-Impala que possui módulos responsáveis pela coleta e análise dos traces de execução, análise da estrutura do programa, classificação e visualização do resultado.

Para avaliar a efetividade, realizamos um estudo utilizando ferramentas diferentes em projetos em manutenção e comparamos os erros (falso-negativos, falso-positivos, revocação e precisão) obtidos por cada uma delas. Em nossos experimentos, a técnica híbrida aumentou a revocação entre 90% e 115% em relação à técnica estática, e aumentou entre 21,2% e 39% em relação à técnica dinâmica. Esses resultados representam que a híbrida aumentou a revocação para ambos os casos e, consequentemente, reduziu o número de falso-negativos.

O restante da dissertação está estruturado da seguinte forma. No Capítulo 2, apresentamos os conceitos sobre análise de impacto necessários para compreender a técnica apresentada neste trabalho. O Capítulo 3 descreve a técnica híbrida de análise de impacto proposta. No Capítulo 5, descrevemos o protótipo de analisador de impacto que implementa a técnica híbrida. O Capítulo 4 descreve o estudo que realizamos para avaliar a técnica e a análise dos resultados. O Capítulo 6 apresenta os trabalhos relacionados seguido pelo Capítulo 7 com as

considerações finais e os trabalhos futuros.

Capítulo 2

Estado da Arte: Análise de Impacto de Mudanças

Neste capítulo, apresentamos os fundamentos teóricos sobre análise de impacto de mudanças necessários para compreensão de nossa solução e estudo. Iniciamos com os conceitos clássicos e técnicas mais conhecidas, finalizando com a apresentação de métricas utilizadas para mensuração dos erros do resultado da análise de impacto durante o estudo para avaliação da técnica híbrida.

2.1 Conceitos Fundamentais

A análise de impacto de mudanças de software, ou simplesmente análise de impacto, identifica as consequências de uma mudança no software [22]. A estimativa errada dos efeitos de uma mudança pode ser considerada responsável pelo alto custo associado às fases de de desenvolvimento e manutenção onde mudanças são requisitadas e realizadas com maior frequência. A análise pode auxiliar no planejamento e estimativa de custos antes da realização da mudança e validar o software mudado em testes de regressão [10; 11].

Na literatura, encontramos diferentes definições para análise de impacto. Para Pfleeger, é a "avaliação dos riscos associados a uma mudança, incluindo a estimativa dos efeitos sobre os recursos, esforço e escalonamento" [20]. Para Richard, análise de impacto determina o escopo de uma mudança e provê uma mensuração de sua complexidade [21]. Já Bohner afirma

que é "'o processo pelo qual é possível determinar as potenciais consequências de uma mudança e estimativa o que será necessário modificar caso uma mudança seja implementada"' [22]. Essa última é a definição utilizada neste trabalho.

A informação fornecida pela análise de impacto pode ser usada para planejar e realizar mudanças, adaptar e configurar o software para as mudanças e localizar os seus efeitos. Além de permitir que os potenciais efeitos da mudança fiquem visíveis antes da sua realização. Outra vantagem na utilização da análise de impacto está em contextos onde há mais de uma alternativa de mudanças. Geralmente, mais de uma mudança pode solucionar um problema ou satisfazer a um mesmo requisito. O conhecimento do impacto causado por cada uma delas é fundamental para decidir qual a melhor alternativa. A informação resultante da análise auxilia engenheiros de software nessa decisão. Em todos esses casos, a análise é aplicada antes da realização da mudança. Mas sua utilização não se restringe a esse contexto. Em atividades de teste de regressão, por exemplo, a análise de impacto determina as entidades do programa que precisam ser testadas novamente depois da realização da mudança.

Geralmente, quando uma mudança é realizada, outras entidades do sistema são afetadas. A identificação dessas entidades é o principal objetivo da análise de impacto. Entretanto, na prática, a análise de impacto identifica entidades "possivelmente" afetadas. Essas entidades não representam exclusivamente as entidades afetadas. Por esse motivo, dizemos que o resultado da análise de impacto apresenta erros [18; 9; 10]. Esses erros correspondem à presença de entidades não impactadas ou ausência de entidades impactadas. São os chamados falso-negativos e falso-positivos, respectivamente.

Suponha uma mudança M em um programa P qualquer. Essa mudança afeta outras entidades de P que compõem o subconjunto A. O resultado da análise de impacto contém entidades consideradas afetadas e é conhecido como conjunto de impacto I. As Figuras 2.1 e 2.2 representam o diagrama de Venn com a relação entre o conjunto I e o conjunto das entidades efetivamente afetadas A pela mudança M no programa P destacando os falso-positivos e os falso-negativos, respectivamente. Na Figura 2.1, o conjunto $\bar{P} = I - R$ representa os falso-positivos do conjunto I. E a Figura 2.2 destaca o conjunto $\bar{N} = R - I$ com os falso-negativos de I.

Minimizar o número de falso-positivos e falso-negativos pode representar o aumento da precisão do resultado da análise de impacto.

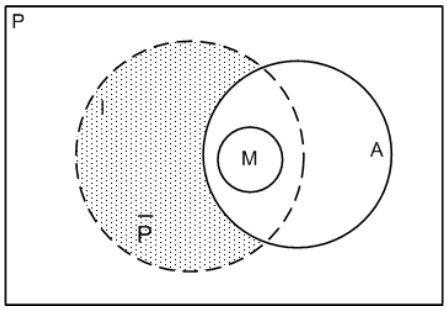


Figura 2.1: Falso-positivos

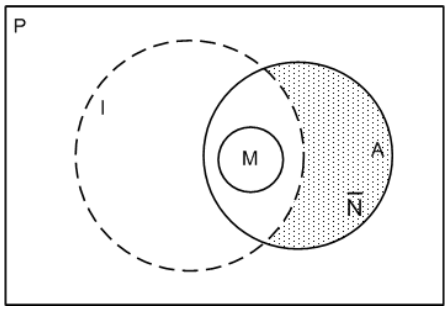


Figura 2.2: Falso-negativos

2.2 Métricas de Análise de Impacto

Neste trabalho, propomos e avaliamos uma técnica híbrida de análise de impacto. Para avaliar a técnica, realizamos um estudo para comparar os erros obtidos por ferramentas diferentes em alguns projetos em manutenção. Para comparar os erros, utilizamos o número falso-positivos e falso-negativos. Entretanto, como são valores absolutos, optamos por utilizar também as métricas precisão e revocação para comparar os resultados entre diferentes projetos. Essas métricas são detalhadas abaixo:

Revocação É uma métrica de recuperação de informação que representa a razão entre o número de documentos relevantes retornados e o número total de documentos relevantes. Essa métrica foi redefinida para o contexto de análise de impacto para representar a razão entre o número de entidades afetadas encontradas pela técnica e o número total de entidades afetadas, ou seja, $\bar{R} = \frac{A \cap I}{A}$ [7]. A revocação é inversamente proporcional ao número de falso-negativos. Portanto, quanto maior for a revocação, menos falso-negativos foram obtidos.

Precisão É uma métrica de recuperação de informação que representa a razão entre o número de documentos relevantes retornados e o número total de documentos retornados por uma consulta. Para o contexto de análise de impacto, essa métrica representa a razão entre o número de entidades afetadas encontradas pela técnica e o número total de entidades obtidas, ou seja, $\bar{P}_r = \frac{A \cap I}{I}$ [7]. A precisão é inversamente proporcional ao número de falso-positivos, ou seja, quanto maior a precisão, menor é o número de falso-positivos.

Considerando que I é o conjunto das entidades estimadas e A é o conjunto das entidades afetadas, temos:

Precisão corresponde é a razão entre o número de entidades corretamente estimadas e o número de entidades estimadas, ou seja:

$$\textit{Precisão ou } P_r = \frac{|A \cap I|}{|I|} \tag{2.1}$$

E revocação é a razão entre o número de entidades corretamente estimadas e o número de entidades afetadas, ou seja:

$$\textit{Revocação ou } R = \frac{|A \cap I|}{|A|} \tag{2.2}$$

Diante disso, para reduzir os erros da análise de impacto, os resultados devem possuir o maior valor possível para revocação e precisão. A Tabela 2.1 apresenta o intervalo de valores que cada uma dessas métricas poderia assumir. Consideramos que o melhor caso para análise aconteceria se o resultado da técnica tiver exclusivamente as entidades afetadas, ou seja, $I = A$. Já o pior caso aconteceria se o resultado da técnica não tiver nenhuma entidade afetada, ou seja, $I \cap A = \phi$.

Tabela 2.1: Intervalo de Valores para as Métricas

Métrica	**Intervalo**
$\bar{N}$	$[0, \|A\|]$
$\bar{P}$	$[0, \|I\|]$
R	$[0, 1]$
P_r	$[0, 1]$

De acordo com a tabela, o número de falso-negativos $\bar{N}$ varia de 0 até o número de entidades efetivamente afetadas pela mudança. O número de falso-positivos $\bar{P}$ varia de 0 até o número de entidades resultantes da análise de impacto. Em ambos os casos, para reduzir os erros, deve-se reduzir seus valores. Para revocação e precisão, os valores variam entre 0 e 1. Nesse caso, para reduzir os erros, esses valores têm que estar mais próximos de 1.

2.3 Técnicas de Análise de Impacto

As técnicas de análise de impacto podem se classificar em duas abordagens: estática ou dinâmica. Na estática, a análise na estrutura do programa define a dependência entre as entidades. Na dinâmica, a análise é realizada nos dados da execução do programa. Estudos

ainda apontam para uma nova tendência: a abordagem híbrida. Essa abordagem consiste na combinação entre a análise da estrutura e dos rastros de execução do sistema.

2.3.1 Análise Estática

A análise de impacto pode utilizar técnicas que utilizam a abordagem estática para identificação do impacto das mudanças. Nesse caso, a análise identifica dependência estrutural entre as entidades analisando artefatos. Classicamente, a análise é realizada em código fonte, mas há trabalhos que analisam outros tipos de artefatos como diagramas UML [4].

Código Fonte 2.1: Pseudocódigo para exemplificar a análise estática

```
1  Classe A
2      m1
3          if x
4              B.m2
5          else
6              C.m3
7
8  Classe B
9      m2
10         y=1
11
12 Classe C
13     m3
14         D.m4
15
16 Classe D
17     m4
18         E.m5
19
20 Classe E
21     m5
22         C.m3
```

São técnicas utilizadas, na maioria das vezes, para compreensão de códigos muito complexos e na depuração de programas. A maioria delas representam os programas em um grafo, onde as entidades são os vértices e os relacionamentos representam as arestas. Dessa

forma, é possível navegar pelo grafo identificando as entidades dependentes da mudança como sendo àquelas alcançáveis a partir do vértice da mudança [8]. A análise ocorre em nível de procedimento, mas pode ser estendível para outros contextos como sistemas orientados a objetos, considerando classes, métodos e atributos como vértices do grafo [15; 7].

Para exemplificar, vamos considerar cinco classes: A, B, C, D e E. Cada classe contém um método que faz uma chamada a um método de outra classe. A Figura 2.3 representa o grafo com as entidades do nosso exemplo. Cada vértice do grafo representa um método e as arestas representam que o método realiza uma chamada para outro.

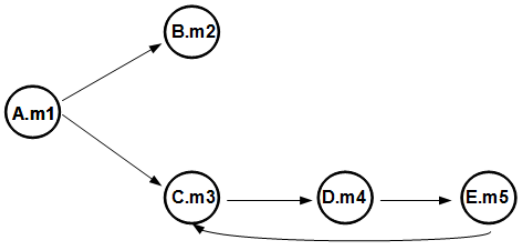


Figura 2.3: Exemplo de grafo de chamadas

Assim, suponha que ocorra uma mudança no método $C.m3$, o conjunto de entidades impactadas seria $\{A.m1, D.m4, E.m5\}$. Isso acontece porque todos esses métodos realizam chamadas diretas ou indiretas para o $C.m3$. Essa situação pode ser vista pela análise no grafo da Figura 2.3, onde os vértices que representam esses métodos podem alcançar o vértice da mudança. A Tabela 2.2 apresenta as relações entre as entidades que devem ser levadas em consideração durante a análise. Como estamos com foco em sistemas orientados a objetos, consideramos que entidades são classes, métodos e atributos. Por exemplo, na primeira linha da tabela, a relação instanceof entre atributo e classe, representa que determinado atributo é uma instância da classe.

Em contextos especiais, como os sistemas orientados a objetos, o grafo gerado pela análise da estrutura é rotulado. Nesse caso, as arestas possuem um rótulo que representam o tipo de relacionamento entre as entidades representadas pelos vértices. A navegação no grafo rotulado vai depender dos rótulos associados a cada aresta. A Tabela 2.2 apresenta as relações entre as entidades que devem ser levadas em consideração durante a análise. Como estamos com foco em sistemas orientados a objetos, consideramos que entidades são

classes, métodos e atributos. Por exemplo, na primeira linha da tabela, a relação *instanceof* entre atributo e classe, representa que determinado atributo é uma instância da classe.

O resultado das técnicas estáticas é conhecido como conjunto de impacto estático. Estudos mostram que o conjunto de impacto estático chega a atingir cerca de 90% do código do programa [2; 1]. Essa situação acontece principalmente por que a análise estática considera todos os comportamentos do sistema, inclusive aqueles impossíveis de acontecer [18]. Por esse motivo, podemos dizer que a maioria das técnicas de análise estática superestima o impacto, incluindo elementos que não são afetados, ou seja, os falso-positivos.

Em alguns contextos, a análise estática pode subestimar o impacto das mudanças. Isso acontece, principalmente, em sistemas orientados a objetos, onde a presença de características especiais alteram a forma de propagação do impacto dificultando a estimativa do escopo da mudança. Nesses sistemas, onde há a possibilidade de utilização de técnicas como *very late binding*, essa situação é ainda mais complicada pois as informações são conhecidas apenas em tempo de execução. Por esse motivo, o resultado da análise estática pode apresentar muitos falso-negativos.

2.3.2 Análise Dinâmica

A abordagem dinâmica para análise de impacto surgiu da necessidade de identificar impactos baseando-se no comportamento do sistema. Pesquisas apontaram que utilizar informação dinâmica na análise representa a obtenção de resultados mais precisos do que a abordagem estática [13; 2; 3].

De maneira geral, a análise é realizada em três etapas: execução do programa analisado, coleta e análise de dados. A execução do programa pode ser realizada utilizando testes ou dados reais de entrada. Os dados da execução são coletados e analisados para identificação de dependência dinâmica entre as entidades. A análise pode ser realizada durante a execução do programa ou após sua finalização. Essas duas formas classificam a análise como *online* e *offline*, respectivamente. O resultado da análise é conhecido como conjunto dinâmico e inclui o subconjunto de entidades do programa que são afetados por uma mudança em pelo menos uma das execuções [10].

As técnicas dinâmicas utilizam heurísticas diferentes para percorrer o rastro de modo a identificar as partes do sistema que dependem da mudança. As técnicas mais conhecidas

Tabela 2.2: Relacionamento entre Entidades

Relacionamento	**Ent. Chamadora**	**Ent. Chamada**	**Descrição**
instanceOf	atributo	classe	Atributo é instância da Classe
contains	classe	atributo, método	Classe contém atributo/método
extends	classe	classe	Subclasse herda características da superclasse
implements	classe	classe	Subclasse implementa todos os métodos da interface
getStatic	método	atributo	Método acessa atributo estático
putStatic	método	atributo	Método modifica atributo estático
getField	método	atributo	Método acessa atributo
putField	método	atributo	Método modifica atributo
invokeVirtual	método	método	Método de uma classe faz chamada a um método de outra classe
invokeSpecial	método	método	Método chama o construtor ou método de uma superclasse ou um método privado
invokeStatic	método	método	Método chama método estático
invokeInterface	método	método	Método de uma classe chama o método de uma interface
is_accessedBy	atributo	método	Atributo é acessado ou modificado por um método, podendo esse tipo de relação ser classificada como: *getStatic*, *putStatic*, *getField* e *putField*
is_invokedBy	atributo	método	Método é invocado pelo outro método. Pode ser classificada como: *invokeVirtual*, *invokeSpecial*, *invokeStatic* e *invokeInterface*
is_superclass	classe	classe	Superclasse é pai de subclasse
catch	método	classe	Método possui em seu corpo um *catch* de exceção do tipo de uma classe
throws	método	classe	Método lança uma exceção do tipo de uma classe
is_declaredOn	método, atributo	classe	Método/atributo é declarado em uma classe *load* método classe método carrega uma classe
is_implementedBy	classe	classe	Superclasse é implementada por subclasse

são PathImpact, CoverageImpact e CollectEA. Para ilustrar a diferença entre essas técnicas, utilizaremos o rastro do exemplo anterior apresentado na Figura 2.4. O rastro consiste na lista de chamadas e retornos de métodos. Para um dado método X, a ocorrência de X_e no rastro indica a chamada de X em um dado ponto da execução. Da mesma forma, uma ocorrência de X_r indica o retorno de X. O rastro da Figura 2.4 representa a execução que A.m1 é chamado.

$$A.m1_e \;\; B.m2_e \;\; C.m3_e \;\; D.m4_e \;\; E.m5_e \;\; E.m5_r \;\; D.m4_r \;\; C.m3_r \;\; B.m2_r \;\; A.m1_r$$

Figura 2.4: Exemplo de Rastro de Execução

O **PathImpact** é uma técnica online que considera as informações em nível de método para realizar a análise. Essas informações são a chamada e o retorno de cada método e a ordem em que aparecem no rastro de execução [13]. Dada uma mudança, PathImpact encaminha e retrocede no rastro para determinar o impacto desta mudança. O encaminhamento à direita determina todos os métodos chamados depois do método mudado, enquanto no sentido inverso identifica os métodos que chamaram o método da mudança. Mais especificamente, para cada método da mudança X e cada ocorrência de X_e:

1. no encaminhamento, PathImpact inicia do sucessor imediato de X_e e inclui todos os métodos chamados após X no conjunto de impacto (por exemplo, todo método Y que contém um evento Y_e após a ocorrência de X_e), e conta o número de eventos de retorno que não apresentam eventos de entrada no mesmo trecho;

2. quando retrocede, o PathImpact inicia a partir do predecessor imediato do X_e e inclui o número de eventos de entrada de acordo com o valor calculado na etapa anterior;

3. finalmente, X é adicionado ao conjunto de impacto;

Considerando o rastro da Figura 2.4 e o método da mudança $C.m3$, temos:

1. Inclusão no conjunto de impacto I os métodos chamados após o método da mudança: I = $\{D.m4, E.m5\}$. E conta o número de métodos não pareados: 2 (B.m2 e A.m1);

2. No sentido inverso, atualizando o I com os 2 eventos imediatamente anteriores ao $C.m3$. Assim, I = $\{A.m1, B.m2, D.m4, E.m5\}$;

3. I = $\{A.m1, B.m2, C.m3, D.m4, E.m5\}$;

CollectEA é a técnica apontada como mais eficiente e mais precisa em análise dinâmica [19]. Baseia-se na relação binária *ExecuteAfter* para identificar as entidades afetadas. Considere duas entidades e_1 e e_2 de um programa P: se e_1 é executado após e_2 em qualquer execução, então $(e_1, e_2) \in EA$ [9; 2]. Segundo Rothermel e Harrold [23], essa relação pode ser definida formalmente como:

Dado um programa P, o conjunto de execuções E e dois métodos X e Y de P, (X,Y) ∈ EA para E, se e somente se, em pelo menos uma execução de E:

1. Y chama X (diretamente ou transitivamente) ou
2. Y retorna em X (diretamente ou transitivamente) ou
3. Y retorna em um método Z (diretamente ou transitivamente), e Z chama X em algum momento posterior (diretamente ou transitivamente).

O CollectEA armazena apenas dois eventos para cada método: a primeira e a última ocorrência do método. A fim de definir quais as relações EA entre os elementos, o algoritmo estabelece que deve ser armazenado o *timeStamp* de cada evento. Esses eventos são ordenados de acordo com o tempo em que foram executados. Essa ordenação permite definir as relações EA entre os métodos.

2.3.3 Análise Híbrida

Embora a maioria das ferramentas e técnicas optarem por utilizar a abordagem estática ou a dinâmica, Em algumas situações, é necessário criar uma nova abordagem para considerar aspectos diferentes na análise. Por esse motivo, surgiu a necessidade de combinar a análise estática e dinâmica e capturar as vantagens de ambas em uma nova abordagem [17].

Em relação a análise de impacto, a bibliografia não apresenta explicitamente uma abordagem híbrida. Porém, seja apresentado como uma técnica dinâmica, o CoverageImpact realiza a análise de impacto hibrida.

O **CoverageImpact** também trabalha em nível de método, mas utiliza cobertura, em vez de rastro, como as informações para calcular impacto conjuntos. A cobertura da informação

relativa a cada execução é armazenada em um vetor que contém bits por método do programa. Se um método é executado, a posição do vetor correspondente é atualizado [18]. No nosso exemplo, a execução do rastro da Figura 2.4 iria produzir o vetor de bit da Tabela 2.3. Como todos os métodos da Código 2.1 foram executados no rastro, então o vetor de bits possui todos os valores ajustados para 1. Porém, caso existisse um método não executado no rastro, seu valor seria 0.

Execução	A.m1	B.m2	C.m3	D.m4	E.m5
Exec1	1	1	1	1	1

Tabela 2.3: Exemplo de Vetor de bits Gerado pelo CoverageImpact

CoverageImpact calcula o impacto conjuntos em duas etapas. Na primeira, usando a informação sobre a cobertura das execuções, identificando as execuções que, em pelo menos uma, o método da mudança foi executado, marcando os métodos que também apareceram nessas execuções. Na segunda, ele calcula a fatia de encaminhamento estático [1] de cada mudança considerando apenas os métodos marcados no passo anterior.

Para ilustrar, considere o exemplo anterior e a análise de impacto da mudança no método $C.m3$. Como $C.m3$ é executado no rastro da Figura 2.4, todos os métodos também executados são marcados, ou seja, MarkedMethods = $\{A.m1, B.m2, D.m4, E.m5\}$. Supondo que a fatia de encaminhamento estática de $C.m3$ deveria incluir todos os métodos do programa, o impacto resultante do seria I = $\{A.m1, B.m2, C.m3, D.m4, E.m5\}$. Como o CoverageImpact desconsidera a ordem dos eventos, o seu resultado pode apresentar erros.

2.4 Considerações Finais

Esse capítulo apresentou os principais conceitos sobre análise de impacto necessários para compreensão da técnica híbrida proposta e do estudo realizado para avaliá-la. Dentre esses conceitos, descrevemos as métricas utilizadas para mensurar os erros da análise de impacto durante a avaliação da técnica híbrida e a apresentação das abordagens estática e dinâmica de análise de impacto.

[1] Do inglês *static forward slice*

Capítulo 3

Técnica Híbrida de Análise de Impacto de Mudanças para Sistemas Orientados a Objetos

Este capítulo tem como objetivo apresentar a técnica híbrida de análise de impacto a priori proposta nesse trabalho. A técnica é híbrida porque engloba técnicas que possuem abordagens diferentes. A nossa solução consiste em realizar a análise de impacto antes da mudança ser implementada levando em consideração aspectos estruturais e dinâmicos para identificação de relacionamento entre as entidades do sistema. Como estamos lidando com sistemas orientados a objetos, consideramos que entidades são classes, métodos e atributos.

A primeira etapa da técnica consiste em utilizar as características inerentes dos sistemas orientados a objetos para identificar relacionamentos entre as demais entidades do sistema com as entidades da mudança. A segunda etapa consiste em analisar o rastro de execução para identificar entidades cujos eventos relacionam-se com os eventos das entidades da mudança. A terceira pode ser dividida em duas outras: combinação dos resultados da análise estática e dinâmica e a ponderação do resultado da técnica híbrida. A ponderação representa atribuir um critério de classificação de acordo com a dependência dinâmica encontrada na segunda etapa e a combinação consiste em unir os resultados das duas etapas anteriores.

Iniciamos este capítulo com uma visão geral da técnica híbrida e seguimos detalhando como cada etapa é realizada. Por fim, apresentamos um exemplo de aplicação da técnica.

3.1 Visão Geral

A técnica híbrida realiza a análise de impacto em três etapas: análise estática da estrutura do sistema, análise de dados da execução do sistema e combinação e ponderação do resultado. O resultado da técnica é uma lista de entidades ponderadas de acordo com a dependência dinâmica.

A Figura 3.1 representa a visão geral da análise de impacto utilizando a técnica híbrida. O engenheiro de software responsável pela análise de impacto desempenha um papel fundamental porque precisa especificar as mudanças, configurar parâmetros que representam a abrangência da análise de impacto e obter os artefatos que serão analisados. Essas responsabilidades correspondem aos três primeiros passos da análise de impacto. Além desses passos, o processo de análise incluia análise estrutural do código fonte, análise dos rastros de execução, combinação e ponderação dos resultados.

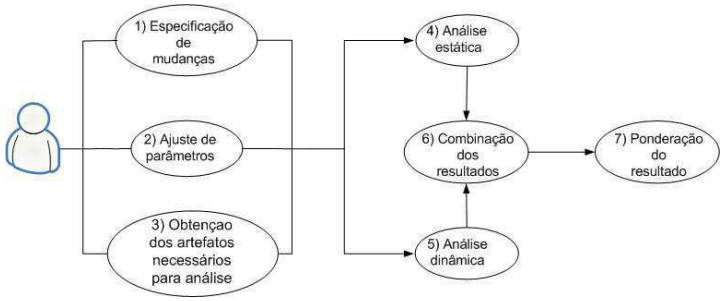


Figura 3.1: Diagrama de use case das ações do engenheiro durante a análise de impacto

Em 1), o engenheiro especifica as mudanças em nível de código. Na Tabela 3.1, cada mudança possui um tipo (alteração, remoção ou inserção) e uma entidade associada (atributo, método ou classe). O tipo representa "o quê" deve ser feito e a entidade representa o "onde". Um exemplo de mudança proposta poderia ser "remover Método b da Classe B" ou "alterar visibilidade Método a da Classe A para privado". As mudanças **inserir** e **remover** representam a requisição da criação e remoção de entidades, respectivamente. Para alterar a visilidade, é preciso dizer qual a nova visibilidade da entidade (*private*, *friendly*, *protected* ou *public*). Para mudanças no corpo do método, consideramos que haverá uma mudança **semântica** neste método. Além disso, é possível inserir ou remover herança em uma classe.

Vamos deixar o ítem 2) para ser explicado durante a apresentação de 5) e 6). Em 3), o en-

Tipo	**Entidade**
Inserir	Classe/Método/Atributo
Remover	Classe/Método/Atributo
Alterar tipo	Atributo
Alterar visibilidade	Classe/Método/Atributo
Renomear	Atributo
Alterar assinatura (Parâmetros, tipo de retorno, exceção lançada)	Método
Alterar semântica	Método
Inserir herança	Classe
Remover herança	Classe

Tabela 3.1: Mudanças analisadas

genheiro deve obter os artefatos necessários para realização da análise de impacto utilizando a técnica híbrida. Esses artefatos são o código fonte e os rastros de execução do programa que deve ser analisado. Em 4), a dependência estrutural das entidades é identificada através da análise do código fonte do programa. Na análise estática que utilizamos na nossa solução, as entidades e os relacionamentos são representados em um grafo e de acordo com as mudanças, é possível navegar e identificar as demais entidades dependentes da mudança. Um dos parâmetros que precisam ser ajustados em 2), é a profundidade. Esse parâmetro representa a profundidade da análise do grafo e, quanto maior for seu valor, mais profunda deve ser a análise estática. Em 5), a análise dos rastros de execução identifica a relação dinâmica entre as demais entidades do sistema e a entidade da mudança. Consideramos que as entidades dependem dinamicamente da mudança se pelo menos um de seus eventos for sucessor do evento da entidade da mudança. O segundo parâmetro que precisa ser ajustado em 2) é a distância de execução. Esse parâmetro representa o número de sucessores que deve ser considerado durante a análise dinâmica.

Em 6), combinamos e ponderamos os resultados obtidos em 4) e 5) de acordo com um fator de impacto que basea-se na dependência dinâmica entre as entidades. Esse fator pode ser usado para auxiliar engenheiros na escolha da melhor alternativa de mudança para o seu projeto. Seu valor varia entre 0 e 1 e representa quão dependente dinamicamente da mudança é a entidade. Quanto mais próximo de 1.0 for o valor do fator de impacto de uma entidade,

mais dependente da mudança ela é. Caso contrário, quando o fator for 0.0, não foi encontrada nenhuma relação dinâmica entre a entidade e a mudança.

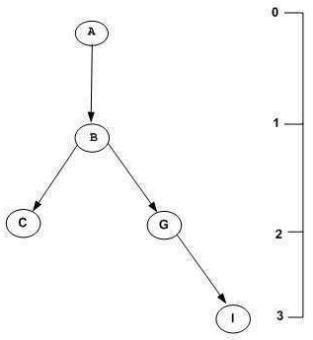
3.2 Etapa 1: Análise Estática

A primeira etapa da análise de impacto realizada pela técnica híbrida consiste na identificação de dependência estrutural entre as demais entidades do sistema e a entidade da mudança através da análise estática. Como o foco da nossa solução é análise de impacto em sistemas orientados a objetos, utilizamos as características inerentes desse tipo de sistemas para determinar a relação entre as entidades.

Na Seção 2.3.1, apresentamos como as entidades se relacionam nesses sistemas. Optamos por utilizar uma técnica de análise de impacto já existente para identificar a dependência estrutural entre as entidades através da análise estática [7]. Essa técnica consiste em analisar o código fonte do programa analisado e gerar um grafo rotulado com informações sobre as entidades e seus relacionamentos. Nesse grafo, os vértices representam as entidades e as arestas o relacionamento entre elas. Então, sempre que um dos relacionamentos da Tabela 2.2 é encontrado, uma aresta ligando os vértices é criada. Cada aresta possui um rótulo que representa o tipo de relacionamento entre as entidades dos vértices.

De acordo com o tipo da mudança, navegamos pelo grafo de modo a encontrar entidades que se relacionam com a entidade da mudança. Essa navegação corresponde a uma busca em *profundidade* por entidades dependentes da mudança e leva em consideração os rótulos associados a cada uma das arestas do grafo. A condição de parada dessa busca é definida pelo parâmetro profundidade e significa quão profunda será a análise. A Figura 3.2 representa um grafo onde A, B, C, G e I são entidades de um programa qualquer e sua profundidade. Quanto maior for a profundidade, maior o número de entidades devem ser considerados na análise.

O algoritmo de navegação depende do tipo de mudança proposta. A Tabela 3.2 apresenta os algoritmos definidos em [14] e aplicados em [7] para análise estática. O algoritmo *Modification* é acionado sempre que uma mudança de remoção de entidade, alteração de tipo e de nome de atributos, alteração de semântica (mudança de escopo do método) e de assinatura de métodos é proposta. Quando é proposta uma mudança em um método, o algoritmo busca os vértices do grafo que possuem o relacionamento do tipo *é chamado por* e retorna os métodos



chamadores; quando for mudança em atributo, busca-se relações *é acessado por* e retorna os métodos chamadores; com mudanças em classe, são retornados os métodos chamadores dos relacionamentos do tipo *é chamado por*.

Algoritmo	**Tipos de Mudanças**
Modification	Remoção de entidades, alteração de tipo e de nome de atributos, alteração de semântica e de assinatura de métodos
ChangeVisibility	Alteração da visibilidade das entidades
RemoveInheritance	Remoção de herança de classe
AddInheritance	Inserção de herança de classe

Tabela 3.2: Algoritmos de navegação utilizados durante a análise estática

Em relação às alterações de visibilidade, quando é uma mudança do menor nível de prioridade para o maior, não há impacto no sistema. Entretanto, quando a mudança é do maior para menor prioridade, muitos impactos são gerados porque alguns objetos podem perder o acesso a dados. Então, o algoritmo leva em consideração as alterações que reduzem a prioridade de visibilidade. O algoritmo *ChangeVisibility* navega no grafo buscando entidades chamadoras que sofrem alteração com aquela mudança. Por exemplo, se a nova visibilidade da entidade for *public*, não há impactos. Entretanto, se uma entidade muda de *public* para *package*, por exemplo, as entidades externas ao pacote perdem o seu acesso e, por consequência, todas as entidades seriam afetadas.

Quando a mudança é do tipo remover herança, os métodos da subclasse que foram herdados pela superclasse são impactados. Dessa forma, entidades que acessam os métodos da superclasse utilizando a subclasse também são impactados. Já na inserção de herança, a

subclasse vai precisar implementar os métodos da superclasse e, por isso, é impactada.

O resultado da técnica estática é um conjunto que inclui as entidades cujos vértices foram visitados durante a navegação pelo grafo. Esse conjunto é combinado com o resultado da análise dinâmica no terceiro passo da análise.

3.3 Etapa 2: Análise Dinâmica

A segunda etapa da técnica híbrida consiste na análise dos rastros de execução para identificação de dependência dinâmica entre as demais entidades do programa e as entidades da mudança. Na nossa solução, consideramos que duas entidades são dependentes dinamicamente se possuem uma relação de sucessão entre seus eventos. Alguns conceitos são fundamentais para compreensão da análise dinâmica proposta em nossa solução e são apresentados a seguir.

3.3.1 Conceitos Fundamentais

O rastro de execução armazena informações sobre a execução do programa. Essas informações são o que chamamos de evento e representam a passagem de uma *thread* por determinado trecho do código do programa em execução. Para nossa solução, nos interessam eventos específicos relacionados a métodos e atributos.

Tipo	**Entidade**	**Identificador**
Entrada	Método	E
Retorno	Método	R
Leitura	Atributo	L
Escrita	Atributo	W

Tabela 3.3: Tipos de eventos

A Tabela 3.3 apresenta os eventos que são levados em consideração durante a análise dinâmica. Os eventos *entrada* e *retorno* representam os eventos de entrada e retorno de um método, respectivamente. Já *leitura* e *escrita* estão associados à leitura e escrita de atributos. Para reduzir o tamanho dos rastros gerados, substituímos eventos sequenciais de entrada e retorno do mesmo método por um meta-tipo de evento: entrada_retorno.

Para nossa solução, entidades são consideradas dependentes dinamicamente se existe uma relação de sucessão entre elas. Essa relação de sucessão é determinada pela ordem dos eventos associados às entidades no rastro. A entidade $ent2$ é dependente de $ent1$ se os eventos de $ent2$ forem sucessores de, pelo menos um, dos eventos de $ent1$. Considere que E^1 é um evento de $ent1$ e E^2 de $ent2$, E^2 é sucessor de E^1 se existe algum trecho do rastro onde E^2 aparece depois de E^1. Utilizando uma linguagem mais formal, dizemos que E^2 é **sucessor** de E^2 se e somente se $\exists R \mid R = r_1\ E^1\ r_2\ E^2\ r_3$, $\forall\ r_1$, r_2 e r_3, onde R é um rastro de execução do programa e r_x é um trecho de R.

Cada sucessor pode ser representado por uma tupla $S = (k, E)$ onde k representa a distância de execução e E o evento. Essa distância de execução é o número de eventos entre os dois que estão sendo analisados. Para o rastro $R = E^1\ r\ E^2\ \forall\ r$, k representa o número de eventos no trecho r. Já o E representa o evento ao qual o sucessor está associado. A tupla que representa a relação de sucessão entre E^2 e E^1 é $S_{E^1} = (2, E^2)$.

A distância de execução k é utilizada para definir um meta-tipo de sucessor: ***k*-sucessor**. Para o rastro R anterior, seja $k = |r|$ então E^2 é dito *k*-sucessor de E^1. Quando $k = 0$, o sucessor é chamado de "sucessor imediato". É interessante observar, que a relação de sucessão e, por conseqüência, de k-sucessão dependem do *rastro* analisado. Para o rastro R do exemplo anterior, E^2 é sucessor de E^1, porém em outro rastro essa situação pode se inverter e E^1 se tornar sucessor de E^2, como em $R' = r_1\ E^2\ E^1\ r_2$.

A relação de sucessão e, por consequência, de *k*-sucessão possui as seguintes características:

- O fato de E^2 ser sucessor de E^1 não implica em E^1 ser sucessor de E^2, ou seja, a relação de sucessão não é simétrica;
- O fato de E^2 ser sucessor de E^1 e E^3 de E^2 implica em E^3 ser sucessor de E^1, ou seja, a relação de sucessão é transitiva;
- Em um mesmo rastro, E^2 pode ser sucessor de E^1 e E^1 de E^2;

3.3.2 Análise dos Rastros de Execução

A segunda etapa da técnica híbrida consiste analisar o rastro de execução e gerar uma estrutura que armazena informações sobre a dependência dinâmica entre as entidades. Dizemos

que as entidades são dependentes dinamicamente se existe uma relação de sucessão entre seus eventos. Portanto, o principal objetivo do analisador dinâmico é identificar os sucessores dos eventos associados às entidades das mudanças.

Obtenção de Rastros

Uma questão que merece atenção na análise dinâmica é a obtenção dos rastros de execução. Para análise de impacto utilizando a técnica híbrida, coletamos as informações dinâmicas através da instrumentação do código fonte. Na nossa solução, o sistema analisado é instrumentado, executado e os dados sobre os eventos são armazenados. Consideramos que os projetos adotam a metodologia de testes automáticos, o que facilita a execução para coleta de informação sobre sua execução. É importante observar que a utilização de testes automáticos não é obrigatória, desde que haja a possibilidade de exercitar o sistema para gerar informações relevantes para análise. Consideramos *informações relevantes* para análise, os rastros de execução que contemplam a dependência entre as entidades da mudança e as demais entidades do sistema.

Definimos uma notação para representação dos eventos como sendo o identificador da entidade indexado com e, r, e_r, l e w para eventos de entrada, retorno e entrada_retorno de método, leitura e escrita de atributo, respectivamente. Alguns exemplos de eventos podem ser: A_e, A_r, F_w e F_l.

Figura 3.2: Exemplo de rastro de execução

Suponha um programa que possui as entidades A, B, C e F, que seu rastro está representado na Figura 3.2. Pretende-se se realizar uma mudança na entidade B. Para identificar os k-sucessores, navegamos pelo rastro buscando pelos eventos de B. Nessa figura, o primeiro evento de B está na posição 2. A partir daí, o índice k é iniciado e os eventos seguintes são armazenados no mapa de sucessores. A medida que os eventos são processados, o mapa é atualizado também com o contador de ocorrência como sucessor. A Figura 3.3 representa o mapa de sucessores gerado após a análise do rastro da Figura 3.2. Para cada evento sucessor, armazenamos um par que representa (k, número de ocorrência). No caso, C_e é 2-sucessor de

B_e uma vez e 0-sucessor duas vezes.

Eventos da Mudança	Sucessores				
	B_R	A_R	C_E	F_W	...
B_E	0,1 2,1 3,1	1,1	2,1 0,2	3,1	...
B_R		...	...	...	...

Figura 3.3: Exemplo de mapa de sucessores

A condição de parada para o processamento dos sucessores é o parâmetro *distância de execução*. Esse parâmetro representa quantos eventos devem ser considerados na análise. Dessa forma, enquanto o índice k for menor que a distância de execução, os sucessores são armazenados no mapa. Esse parametrização permite que o usuário configure quão abrangente deve ser a análise dinâmica.

O resultado da análise dinâmica é o mapa contendo os sucessores e o número de ocorrência dos eventos. Essas informações são combinadas com o conjunto produzido na etapa apresentada na Seção 3.2. Essa combinação dos resultados é apresentada a seguir.

3.4 Etapa 3: Combinação dos Resultados

A última etapa da análise de impacto híbrida consiste em combinar os resultados obtidos pelas etapas anteriores e classificá-los de acordo com as informações dinâmicas. Essas informações dinâmicas nos permitem definir um fator de impacto que representa quão dependente das entidades da mudança são as entidades obtidas nos resultados estático e dinâmico. Esse critério de classificação é calculado a partir das probabilidades dos eventos das entidades serem sucessoras dos eventos das entidades da mudança. Entretanto, para calcular essas probabilidades é necessário eliminar algumas informações irrelevantes para análise. Por isso, definimos um limiar de relevância para auxiliar na identificação das informações relevantes para análise.

3.4.1 Limiar de Relevância

Na maioria das vezes, o mapa de sucessores contém muita informação, algumas delas irrelevantes para análise. Consideramos que essas informações irrelevantes são "ruídos" e, consequentemente, podem ser desconsiderados no cálculo do fator de impacto porque podem confundir o engenheiro de software durante a análise de impacto. Para reduzir a quantidade desse tipo de informação, definimos um limite inferior para ocorrência eventos. Assim, o início da última etapa da análise é definir esse limitar de relevância.

A Figura 3.4 é um gráfico com a contagem da relação entre dois métodos A e B de um programa analisado durante a proposta de nosso trabalho. O gráfico representa o número de ocorrência de B como k-sucessor de A. Como pode ser visto, a ocorrência varia de 9 a 17 para 1; 3; 5; 7 e 9-sucessor e do 21 ao 91-sucessor, a ocorrência é muito inferior, variando entre 1 e 2. Para nossa solução, o limiar de relevância é calculada de forma ingênua como sendo a média das ocorrências de todos os sucessores. Assim, desconsideramos os k-sucessores cuja ocorrência é menor que a média. Na Figura 3.4, o limiar de relevância dos k-sucessores é 7,5 e está representando pela linha vermelha.

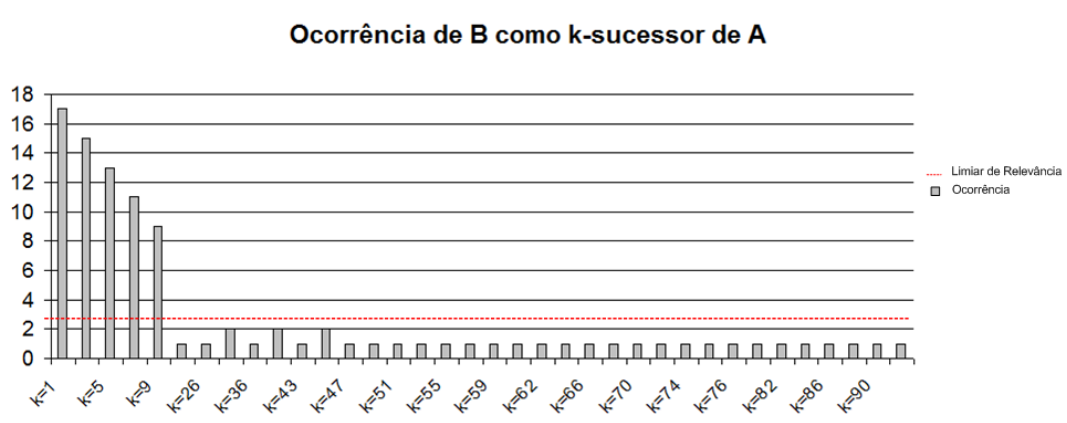


Figura 3.4: Relação entre dois métodos e o limiar de relevância

3.4.2 Cálculo das Probabilidades de Execução

Como os engenheiros precisam analisar uma quantidade muito grande de informação, propomos a utilização de um critério de classificação chamado de fator de impacto. Esse fator é

uma informação adicional que auxilia na organização dos resultados fornecidos pela técnica híbrida. É calculado a partir das probabilidades de ocorrência dos eventos como sucessores. Assim, após selecionar as informações relevantes para análise, calculamos os valores das seguintes probabilidades para todas as entidades obtidas nas etapas 1 e 2:

1. Probabilidade da entidade ter um evento sucessor dos eventos das entidades da mudança;

2. Probabilidade da entidade não ter um evento sucessor dos eventos das entidades da mudança;

Para calcular a primeira probabilidade, consideramos a ocorrência do evento da entidade da mudança no rastro e a ocorrência que do evento analisado como seu sucessor. A variável $ocorrencia(sucessor)$ representa o número de vezes que o evento foi sucessor dos eventos da mudança e $ocorrenciaTotal(mudanca)$ representa o número total de vezes que os eventos da mudança aparecem no rastro. Dessa forma, a probabilidade do evento ser sucessor é calculado como:

$$pSucessor = \frac{ocorrenciaDoEventoSerSucessor}{ocorrenciaTotalDoEventoDaMudanca} \tag{3.1}$$

O passo seguinte é o cálculo da probabilidade do evento não ser sucessor dos eventos da mudança. Esse cálculo segue a seguinte função:

$$pNaoSucessor = \frac{ocorrenciaTotalDoEvento - ocorrenciaDoEventoSerSucessor}{ocorrenciaTotalDoEventoDaMudanca} \tag{3.2}$$

A variável $ocorrenciaTotal(evento)$ representa a quantidade de vezes que o evento aparece no rastro e $ocorrencia(sucessor)$ representa quantas vezes o evento foi sucessor dos eventos da mudança.

3.4.3 Cálculo do Fator de Impacto

O fator de impacto das entidades é calculado por uma combinação entre essas duas probabilidades:

$$fatorDeImpactoDoEvento = \frac{pSucessor}{pSucessor + pNaoSucessor} \tag{3.3}$$

É importante observar que o fator de impacto é calculado de acordo com as probabilidades dos *eventos* serem sucessores. Porém, o fator de impacto que nos interessa está relacionado às entidades que pertencem nos resultados estático e dinâmico. Como a mesma entidade pode estar associada a mais de um evento diferentes, por exemplo um método que possui entrada e retorno, estabelecemos que prevalece o maior de impacto.

Para atributos, o cálculo do impacto é realizado de forma diferente. A variável $ocorrenciaTotalEvento$ é a soma das ocorrências, consequentemente fatorDeImpacto dos eventos de leitura e escrita de atributos são iguais.

3.4.4 Combinação de Resultados

Todas as entidades resultantes da análise estática e da dinâmica devem ter um fator de impacto associado. O código 3.1 apresenta o algoritmo que combina os resultados. As linhas de 4 a 6 correspondem a navegação pelos elementos do mapa de sucessão e a linha 7 corresponde ao cálculo do fator de impacto. Após isso, navega-se pelas entidades obtidas pela análise estática (linha 10), se ainda não pertence a lista resultante, calcula-se seu fator de impacto (linha 12) e as insere na lista de impacto (linha 12 e 13).

Código Fonte 3.1: Algoritmo de combinação dos resultados

```
1  Algoritmo CombinaResultados(Conjunto estatico, MapaDeSucessores mapa)
2  Inicio
3          List listaDeImpacto
4          Para cada mudanca em mapa.getMudancas()
5                  List sucessores = mapa.getSucessores(mudanca)
6                  Para cada sucessor em sucessores
7                          fatorDeImpacto = CalculaFatorDeImpacto(sucessor.
                               getEntidade())
8                          listaDeImpacto.add(sucessor.getEvento().
                               getEntidade(), fatorDeImpacto)
9
10         Para cada entidade em estatico
11                 Se !listaDeImpacto.contem(entidade)
```

```
12                          fatorDeImpacto = CalculaFatorDeImpacto(entidade)
13                          listaDeImpacto.add(entidade, fatorDeImpacto)
14  Fim
```

3.5 Exemplo de Aplicação da Técnica

Para exemplificar a aplicação da técnica híbrida de análise de impacto, utilizaremos o sistema ToyExample cujo diagrama de classes está apresentado na Figura 3.5. Iremos realizar a análise de impacto das seguintes mudanças nesse sistema:

1. Alterar assinatura do método example.command.CommandWithLabel.getLabel() para example.command.CommandWithLabel.getLabel()
2. Remover o método example.app.Application.addmenuItem(example.command.CommandWithLabel)
3. Alterar a assinatura do método example.app.Application.addMenuItem(String) para example.app.Application.addMenuItem(int)

É importante relembrar que a técnica híbrida é representada por 3 etapas: análise estática, dinâmica e combinação de resultados. Na **primeira etapa**, identificamos a dependência estrutural entre as entidades utilizando a análise estática. Durante a análise estática, geramos um grafo de entidades e relacionamentos representando vértices e grafos. Considerando a mudança 1, a técnica consiste em identificar, pela análise da estrutura do código fonte, as entidades que se relacionam com a entidade que será mudada.

A Tabela 3.4 representa o resultado da primeira etapa realizada através da navegação pelo grafo gerado durante a análise estática. Como pode ser visto, o resultado dessa etapa forma um conjunto de entidades consideradas afetadas.

A **segunda etapa** consiste em identificar os sucessores dos eventos da mudança e geração do mapa de sucessores. O rastro do ToyExample possui 1386 eventos e sua análise gera um mapa de sucessores com 216 elementos. Seria inviável colocá-los nesse trabalho, por isso selecionamos alguns elementos para exemplificar a análise dinâmica da técnica híbrida.

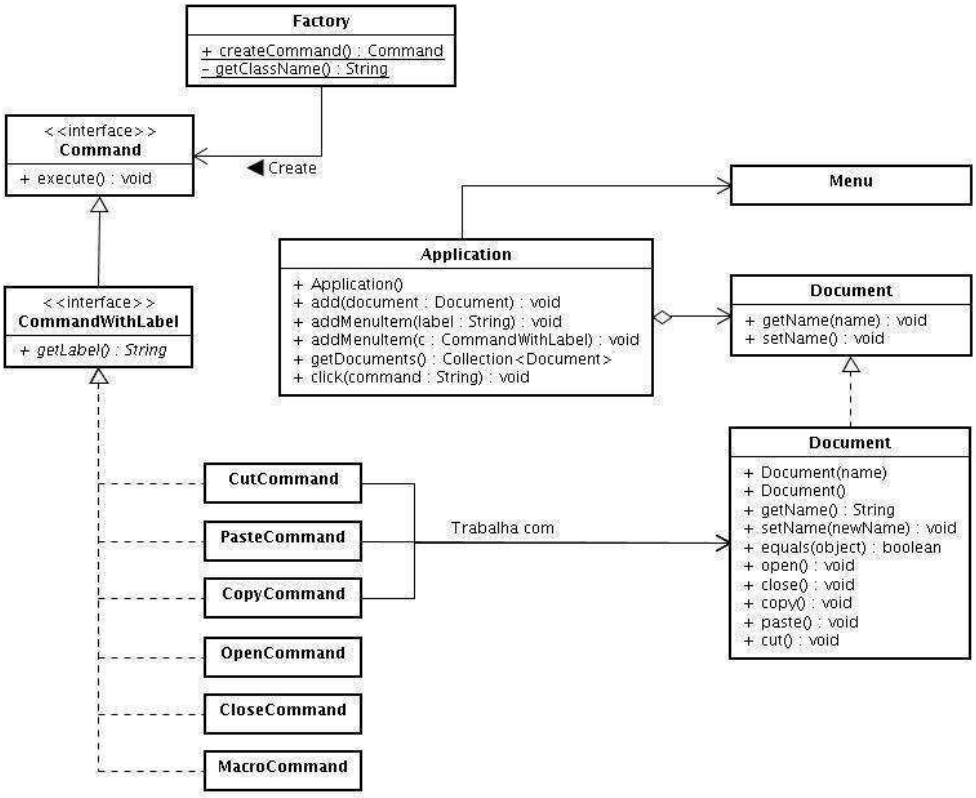


Figura 3.5: Diagrama de classes do ToyExample

Entidade da mudança	Sucessores
Application.addMenuItem(String)	Document.<init>(java.lang.String)$_R$ é 3-sucessor 1 vez Document.open()$_E$ é 9-sucessor, 14-sucessor, 62-sucessor, 165-sucessor, 199-sucessor e 234-sucessor 1 vez Document.open()$_R$ é 11-sucessor, 16-sucessor, 64-sucessor, 167-sucessor e 236-sucessor 1 vez
Application.addMenuItem(CommandWithLabel)	PasteCommand.<init>(example.app.Document)$_E$ é 180-sucessor 1 vez PasteCommand.<init>(example.app.Document) $_R$ é 183-sucesspr 1 vez

Figura 3.6: Mapa de sucessores do ToyExample

Entidades
example.command.CopyCommand.<init>(example.app.Document)
example.app.Application.addMenuItem(java.lang.String)
example.command.MacroCommand.<init>()
example.command.CopyCommand.<init>()
example.tests.CloseCommandTest.testCloseCommand()
example.tests.TestApplication.testCreateMenuItem()
example.tests.TestApplication.testClick()
example.tests.TestApplication.testCreateMenuItem1()
example.command.CommandWithLabel.getLabel()
example.command.CommandWithLabel
example.command.CloseCommand
example.command.OpenCommand.<init>()
example.command.PasteCommand.<init>()
example.command.PasteCommand
example.command.MacroCommand
example.command.CopyCommand
example.command.CutCommand.<init>()
example.tests.CommandTest.testCloseCommand()
example.command.PasteCommand.<init>(example.app.Document)
example.command.CutCommand
example.command.CloseCommand.<init>()
example.app.Application.addMenuItem(example.command.CommandWithLabel)
example.tests.CommandTest.testCopy()
example.command.CutCommand.<init>(example.app.Document)
example.tests.TestMacroCommand.testGetLabel()

Tabela 3.4: Exemplo do resultado produzido durante a análise estática

A Figura 3.6 representa uma parte do mapa de sucessores gerados após a análise do rastro do ToyExample. Como podemos observar, o mapa possui os sucessores dos eventos associados a entidade da mudança e o número de ocorrências dessa relação. Cada entidade da mudança possui uma lista de sucessores e sua respectiva ocorrência. Por exemplo, o evento Document.<init>(java.lang.String)$_R$ aparece como 3-sucessor dos eventos de Application.addMenuItem(java.lang.String) uma única vez no rastro.

Na **terceira etapa**, as informações do mapa de sucessores são utilizadas para ponderar o conjunto estático e o resultado dinâmico. Essa ponderação ocorre através de um critério de classificação que chamamos de fator de impacto. Essa fator de impacto poderá auxiliar engenheiros na organização da informação produzida pela análise. Para exemmplificar, calculamos o fator de impacto do Document.open(). A seguir, disponibilizamos algumas informações indispensáveis para o cálculo do fator de impacto:

1. ocorrenciaTotalDaMudanca = ocorrencia(Application.addMenuItem(java.lang.String)$_E$) + ocorrencia(Application.addMenuItem(String)R) = 10;

2. ocorrenciaTotalDoEvento = ocorrenciaTotal(example.app.Document.open()$_E$) = ocorrenciaTotal(example.app.Document.open()$_R$) = 10;

3. example.app.Document.open()$_E$ é 9-sucessor, 14-sucessor, 62-sucessor, 165-sucessor, 199-sucessor e 234-sucessor 1 vez cada;

4. example.app.Document.open()$_R$ é 11-sucessor, 16-sucessor, 64-sucessor, 167-sucessor, 201-sucessor e 236-sucessor 1 vez cada;

O primeiro passo para cálculo do fator de impacto é determinar o limiar de relevância representado pela média de ocorrências dos sucessores: Media = $\frac{1+1+1+1+1+1}{6}$ = 1. Nesse caso, todas as ocorrências dos sucessores devem ser levados em consideração no cálculo do fator de impacto porque não são inferiores à média das ocorrências.

O segundo passo é calcular o valor das probabilidades 3.1 e 3.2 para os eventos de Document.open() que são Document.open()$_E$ e Document.open()$_R$. Para Document.open()$_E$ e Document.open()$_R$, temos: pSucessor = $\frac{6}{10}$ = 0,6 e pNaoSucessor = $\frac{10-6}{10}$ = 0,4.

Por fim, o cálculo do fator de impacto de acordo com a Função 3.3: fatorDeImpacto(Document.open()$_E$) = $\frac{0,6}{0,6+0,4}$ = 0,6 e fatorDeImpacto(Document.open()$_R$) =

0,6. O fator de impacto de Document.open() = Max(Document.open()$_E$, Document.open()$_R$) = 0,6.

3.6 Considerações Finais

A técnica de análise híbrida elaborada neste trabalho tem como objetivo principal reduzir os falso-negativos das técnicas que utilizam a abordagem tradicional. A análise de impacto utilizando a técnica híbrida é realizada em 3 etapas: análise estática, análise dinâmica e combinação dos resultados. O primeiro passo consiste na análise estática para identificação de dependência estrutural entre as demais entidades do sistema e as entidades das mudanças. A segunda corresponde a identificação de dependência dinâmica baseada na relação de sucessão entre os eventos das entidas do sistema e as entidades da mudança. A última consiste na combinação dos resultados das etapas anteriores e sua classificação baseada nas informações dinâmicas.

Nesse capítulo, apresentamos os detalhes sobre as etapas da solução proposta. A primeira etapa da técnica consiste em analisar o relacionamento entre entidades baseada em características de sistemas orientados a objetos. Utilizamos uma técnica definida em um trabalho anterior que representa entidades e relacionamentos como vértices e arestas de um grafo e que possui um parâmero *profundidade* que representa o alcance da análise nesse grafo. A segunda etapa consiste em coleta e análise dos dados da execução do programa e identificação dos sucessores relacionados às entidades das mudanças. Por fim, as informações sobre o número de ocorrência que os eventos como sucessores são utilizadas para classificar os resultados obtidos nas etapas anteriores.

Por fim, apresentamos um exemplo de como a técnica pode ser aplicada.

Capítulo 4

Avaliação da Técnica de Análise de Impacto

Neste capítulo, apresentamos o estudo realizado para verificar a efetividade da técnica híbrida de análise de impacto proposta. O estudo consistiu na avaliação da hipótese de que a abordagem híbrida de análise de impacto permite reduzir a quantidade de falso-negativos obtidos em relação às técnicas estática e dinâmica existentes.

No estudo, selecionamos alguns projetos e realizamos a análise de impacto utilizando ferramentas diferentes e comparamos os erros obtidos por cada uma delas. Para compreender o que são esses erros da análise, revisaremos os conceitos de falso-negativos, falso-positivos, revocação e precisão.

4.1 Conceitos Fundamentais

Antes de compararmos os resultados da análise de impacto, apresentaremos os conceitos de falso-positivos e falso-negativos. Para isso, dividimos as entidades em dois grupos: entidades efetivamente mudadas e entidades estimadas. O primeiro grupo contém as entidades afetadas pela mudança e o segundo contém as entidades obtidas pela ferramenta de análise de impacto. A Figura 4.1 apresenta a relação entre esses dois grupos. Se o resultado da ferramenta possui uma entidade e esta não foi afetada então a análise apresenta um erro chamado de falso-positivo. No caso inverso, onde o resultado da ferramenta não possui a entidade e esta foi afetada então a análise apresenta um erro chamado de falso-negativo. Se a entidade

foi afetada e encontrada pela ferramenta ou não foi afetada e não foi estimada, a análise está correta [16].

		Entidade	
		Não estimada	Estimada
Entidade	Afetada	Erro (falso-negativo)	Correto
	Não afetada	Correto	Erro (falso-positivo)

Figura 4.1: Relação entre entidades afetadas e obtidas

Assim, consideramos que falso-positivos são as entidades contidas no resultado da análise que não são afetadas pela mudança e falso-negativos são entidades afetadas e que não foram obtidas pela análise. Consideramos que os falso-positivos e falso-negativos correspondem ao erro da análise.

Entretanto, utilizar o número de falso-positivos e falso-negativos para mensurar os erros da análise é insuficiente porque são absolutos e não podemos comparar seus valores em contextos diferentes. Por esse motivo, também utilizamos as métricas precisão e revocação. Essas métricas são aplicadas para recuperação de informação, mas foram redefinidas para o contexto da análise de impacto [7]. A revocação é a razão entre as entidades afetadas obtidas pela técnica e o número total de entidades afetadas. Já a precisão é a razão entre as entidades afetadas obtidas pela técnica e o número total de entidades obtidas. Assumimos que o número de falso-negativos é inversamente proporcional à revocação e o número de falso-positivos é inversamente proporcional à precisão.

4.2 Caracterização das questões de pesquisa

Para avaliar a efetividade da técnica híbrida, realizamos um estudo para validar a hipótese de que a abordagem híbrida de análise de impacto permite reduzir a quantidade de falso-negativos obtidos em relação às técnicas estática e dinâmica existentes. Formalmente, essa hipótese pode ser expressa por:

$$\bar{N}_{hibrida} < \bar{N}_{estatica} \text{ e } \bar{N}_{hibrida} < \bar{N}_{dinamica}$$

Onde: $\bar{N}_{hibrida}$, $\bar{N}_{estatica}$ e $\bar{N}_{dinamica}$ representam os falso-negativos obtidos pelas técnicas híbrida, estática e dinâmica, respectivamente.

Como consideramos que a revocação é inversamente proporcional ao número de falso-negativos, podemos acrescentar à hipótese a seguinte condição:

$$R_{hibrida} > R_{estatica} \text{ e } R_{hibrida} > R_{dinamica}$$

Onde: $R_{hibrida}$, $R_{estatica}$ e $R_{dinamica}$ representam os valores da revocação das técnicas híbrida, estática e dinâmica, respectivamente.

Assim, optamos por decompor a hipótese em duas para facilitar a compreensão do resultado do nosso estudo.

H1: $\bar{N}_{Hibrida} < \bar{N}_{Estatica} \text{ e } R_{Hibrida} > R_{Estatica}$

H2: $\bar{N}_{Hibrida} < \bar{N}_{Dinamica} \text{ e } R_{Hibrida} > R_{Dinamica}$

A seguir, apresentamos os materiais e métodos utilizados na validação da hipótese.

4.3 Materiais e Métodos

Para realizar esse estudo, utilizamos os projetos em manutenção, mudanças previstas para suas próximas versões e ferramentas que implementem a técnica híbrida e as técnicas (dinâmica e estática) de controle.

4.3.1 Seleção de Projetos

Os projetos foram selecionados, observando-se os seguintes requisitos:

- O **projeto** deveria se encontrar **em evolução**. Isto nos permite identificar mudanças realistas que estejam sendo realizadas no software;
- **Acesso ao repositório do código** para análise de duas versões diferentes (antes e depois da mudança). Com esse acesso, é possível realizar a análise estática do código fonte e a dinâmica dos dados sobre os rastros de execução do projeto;

- **Contato e disponibilidade do responsável** pelo projeto para colaborar com a implementação das mudanças durante o tempo de realização de nossos experimentos;

Selecionamos 4 projetos desenvolvidos pelo Grupo de Métodos Formais que atendem a esses requisitos: Design Wizard[1], Design[2], Abstractor[3] e o Impala[4]. Além desses projetos, desenvolvemos um protótipo "'brinquedo"' chamado ToyExample que também foi utilizado nos experimentos. A Tabela 4.1 apresenta os sistemas selecionados para o estudo e algumas de suas medidas características como o LOC (line of code), número de classes e o nível de cobertura dos testes. Os projetos possuem testes automáticos e com nível de cobertura superior a 52%, além disso, são projetos relativamente pequenos (LOC entre 678 e 5969) porque não conseguimos obter projetos grandes que atendessem a todos os requisitos apresentados anteriormente.

Projeto	**LOC**	**Núm. de Classes**	**Nível de Cobertura de Testes**
DesignWizard	5825	67	57,2%
Design	2520	18	52,49%
Abstractor	5969	77	64,80%
Impala	5176	57	63,63%
ToyExample	678	20	84,22%

Tabela 4.1: Projetos Selecionados e Algumas Métricas

4.3.2 Isolamento de Mudanças

O isolamento das mudanças nos permite obter o conjunto da mudança (M) e o de entidades afetadas (A). Para obter o M, identificamos dentre as mudanças previstas dos projetos aquelas que se enquadravam no contexto de nossos experimentos: 1) testes deveriam cobrir trecho do código da mudança e 2) não poderiam ser do tipo “inserir entidade”. Esse primeiro requisito deve ser atendido para que o rastro de execução gerado consiga capturar informações relevantes para análise dinâmica. Já o segundo, é importante porque o impacto de “inserir

[1] API de inspeção em códigos Java que extrai informações de design a partir do *byte code* de programas [5]

[2] Protótipo que gera um grafo para extrair informações básicas sobre o design

[3] API para operações de abstração sobre um design de software

[4] Analisador de impacto que utiliza dados evolutivos do software [7]

uma entidade" é a própria mudança e seria irrelevante para o resultado de nossos experimentos. O tamanho desse conjunto variou entre 3 e 7 elementos e correspondiam a mudanças do remoção de entidade, alteração de assinatura e de semântica. Essas mudanças foram implementadas e uma versão estável foi gerada (testes devem estar passando corretamente).

Para obter o conjunto de entidades afetadas A, isolamos duas versões do software: uma anterior e outra posterior à mudança. E assumimos que esse conjunto A possui entidades alteradas durante a realização das mudanças e que são encontradas pela diferença entre as duas versões.

4.3.3 Obtenção das Ferramentas de Análise de Impacto

Dado que temos o conjunto de mudanças M e as entidades afetadas A, precisamos obter o conjunto de impacto de cada técnica $I_{estatico}$, $I_{dinamico}$ e $I_{hibrido}$ para o cálculo e comparação das métricas número de falso-negativo ($\bar{N}$) e revocação (R). Dessa forma, para cada um dos projetos, realizamos a análise de impacto das mudanças utilizando ferramentas que implementam as técnicas estática, dinâmica e híbrida. Escolhemos as ferramentas: Impala que representa a análise estática; CollectEA como representante da análise dinâmica e o *SD*-Impala como o analisador híbrido.

O **CollectEA** é a técnica dinâmica mais eficiente e precisa [2]. A análise de impacto é baseada na relação boleana *executeAfter* que pode ser definida formalmente como:

Dado um programa P, um conjunto de rastros de execução E, e dois métodos X e Y de P, (X, Y) ∈ executeAfter para E se e somente se, em pelo menos um dos rastros de E:

1. Y chama X (diretamente ou transitivamente),
2. Y retorna em X (diretamente ou transitivamente), ou
3. Y retorna em um método Z (diretamente ou transitivamente), e o método Z depois chama X (diretamente ou transitivamente).

Entretanto, não encontramos uma ferramenta que implemente a técnica. Por esse motivo, implementamos a nossa própria versão do CollectEA para representar a ferramenta de controle da análise dinâmica.

Conforme apresentado na Seção 5.2, o **Impala** é um analisador de impacto que considera as características de orientação a objetos para identificação do impacto [7]. O protótipo anal-

isa o código fonte do programa e identifica dependências estruturais entre as entidades. Sua idéia principal consiste em representar a estrutura do programa em um grafo onde vértices e arestas representam entidades e seus relacionamentos. De acordo com a mudança proposta, navega pelo grafo realizando uma busca em profundidade a fim de identificar quais as entidades dependentes da mudança. Possui o parâmetro "profundidade" como a condição de parada e representa quão abrangente deve ser a análise. Optamos por utilizá-lo como a ferramenta de controle da técnica estática por que se aplicaria adequadamente ao nosso estudo pela facilidade de acesso.

O **SD-Impala** é o protótipo que representa a técnica híbrida elaborada neste trabalho. Sua arquitetura está descrita no Capítulo 5 e, assim como na técnica apresentada no Capítulo 3, requer a passagem de dois parâmetros: profundidade e distância de execução. O primeiro parâmetro representa quão abrangente será a análise estrutural e o segundo representa o número de sucessores que devem ser considerados durante a análise das informações dinâmicas.

Ferramenta	**Técnica**	**Parâmetros**
CollectEA	Dinâmica	-
Impala	Estática	Profundidade
SD-Impala	Híbrida	Profundidade e Distância de Execução

Tabela 4.2: Ferramentas de Análise de Impacto escolhidas

A Tabela 4.2 apresenta as ferramentas utilizadas para o nosso estudo, as técnicas que implementam e os parâmetros necessários pra realização da análise de impacto. Esses são parâmetros específicos de cada ferramenta, mas existem os parâmetros gerais para as ferramentas de análise de impacto: as propostas de mudanças e o programa analisado.

A análise de impacto nos projetos

Para realizarmos a análise de impacto utilizando as ferramentas citadas acima, foi necessário:

1. Coletar os rastros de execução com os eventos apresentados na Seção 5.3.1;
2. Definir as mudanças selecionadas utilizando a sintaxe da *Change Description Language* apresentada na Seção 5.1;

3. Realizar a análise de impacto utilizando cada uma das ferramentas escolhidas. Antes de realizar esse passo, tivemos que definir quais os valores que os parâmetros profundidade e distância de execução deveriam assumir no nosso estudo. Para isso, realizamos experimentos prévios e verificamos que, na maioria das situações, o resultado da análise estática sofria alteração quando variamos o valor da profundidade em 5, 10 e 20. O mesmo acontecia para o resultado da análise dinâmica quando a distância de execução variava em 100, 500 e 1000. Como o CollectEA não é uma técnica parametrizável, realizamos a análise de impacto variando apenas os parâmetros do Impala e do SD-Impala. Para o Impala, variamos a profundidade em valores baixo (5), médio (10) e alto (20). Enquanto para o SD-Impala, a profundidade recebeu os mesmos valores do Impala (5, 10 e 20), enquanto a distância de execução recebeu valores baixo (100), médio (500) e alto (1000);

Assim, obtivemos os conjuntos I de cada ferramenta e calculamos os falso-positivos, falso-negativos, precisão e revocação. A seção a seguir apresenta esses valores.

4.4 Resultados

Dividimos os resultados dos experimentos de acordo com os parâmetros utilizados para realização da análise de impacto com as ferramentas. Assim, classificamos em três grupos: (i) profundidade = 5 e distância de execução=100, (ii) profundidade = 10 e distância de execução=500 e (iii) profundidade = 20 e distância de execução=1000. Dessa forma, apresentamos a seguir os valores mínimo, máximo e a média dos falso-negativos e revocação para cada uma das ferramentas. O resultado completo da análise encontra-se no Apêndice A.

4.4.1 Falso-negativos e Revocação

A Tabela 4.3 apresenta o número de falso-negativos e a revocação para profundidade = 5 e distância de execução = 100. Nessa tabela, apresentamos os valores mínimo, máximo e a média obtidos para essas duas métricas. Para relembrar nossa hipótese, os valores do SD-Impala do falso-negativos deve ser menor que os das demais ferramentas e a revocação deve

ser maior.

Profundidade = 5 e Distância de execução = 100						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{N}$	R	$\bar{N}$	R	$\bar{N}$	R
CollectEA	3	0,092	22,2	0,423	59	0,722
Impala	6	0,111	19,6	0,269	27	0,631
SD-Impala	3	0,375	14,8	0,513	22	0,677

Tabela 4.3: Falso-negativos e revocação para o cenário (i)

De acordo com os valores da Tabela 4.3, observamos que no valor mínimo e máximo nossas expectativas não foram atendidas em dois valores para o CollectEA. Isso aconteceu porque para o número de falso-negativos mínimo, o SD-Impala obteve valor igual ao do CollectEA e a revocação máxima foi menor ($\bar{N}_{SD-Impala}$ = $\bar{N}_{CollectEA}$ e $R_{SD-Impala}$ < $R_{CollectEA}$). Os demais valores estão de acordo com a hipótese investigada pois obtivemos pois obtivemos $\bar{N}_{SD-Impala}$ < $\bar{N}_{CollectEA}$ e $\bar{N}_{SD-Impala}$ < $\bar{N}_{Impala}$ e $R_{SD-Impala}$ > R_{Impala}.

Para o cenário (ii) onde profundidade = 10 e distância = 500, os valores dos falsonegativos estão apresentados na Tabela 4.4. Nesse cenário, nossas expectativas foram atendidas em todos os valores porque o número de falso-negativos do SD-Impala foi sempre menor que os demais e a revocação foi sempre maior.

Profundidade = 10 e Distância de execução = 500						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{N}$	R	$\bar{N}$	R	$\bar{N}$	R
CollectEA	3	0,092	22,2	0,423	59	0,722
Impala	6	0,111	19,4	0,272	27	0,646
SD-Impala	2	0,389	13	0,586	22	0,714

Tabela 4.4: Falso-negativos e revocação para o cenário (ii)

Para o cenário (iii) que contempla a análise mais abrangente, onde profundidade = 20 e distância = 1000, os valores dos falso-negativos estão apresentados na Tabela 4.5. Nesse cenário, as nossas expectativas também foram atendidas pois para todos os menores valores de falsonegativos e os maiores valores da revocação foram obtidos pelo SD-Impala.

Profundidade = 10 e Distância de execução = 500						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{N}$	R	$\bar{N}$	R	$\bar{N}$	R
CollectEA	3	0,092	22,2	0,423	59	0,722
Impala	6	0,111	19,2	0,275	27	0,662
SD-Impala	2	0,389	12,8	0,588	22	0,723

Tabela 4.5: Falso-negativos e revocação para o cenário (iii)

A partir da análise das tabelas apresentadas acima, podemos dizer que em apenas um cenário os valores não atenderam nossas expectativas porque o SD-Impala obteve o número dos falso-negativos mínimo igual e a revocação máxima menor do que os valores do CollectEA.

4.4.2 Falso-positivos e Precisão

Embora não seja parte de nossa hipótese, coletamos os valores dos falso-positivos e precisão para verificar o comportamento do SD-Impala em relação às demais ferramentas. Com isso, podemos estabelecer os limites da nossa técnica, também, em relação aos falso-positivos.

Para garantir que nossa técnica reduz os erros relacionados aos falso-positivos, então as seguintes condições devem ser atendidas:

C1: $\bar{P}_{SD-Impala} < \bar{P}_{Impala}$ e $Pr_{SD-Impala} > Pr_{Impala}$

C2: $\bar{P}_{SD-Impala} < \bar{N}_{CollectEA}$ e $Pr_{SD-Impala} > Pr_{CollectEA}$

As Tabelas 4.6, 4.7 e 4.8 apresentam os valores mínimo, máximo e a média dos falso-positivos e da precisão obtidos pelas ferramentas com os parâmetros do grupo (i), (ii) e (iii), respectivamente. Em nenhum dos contextos, a condição C1 foi atendida, ou seja, $\bar{P}_{SD-Impala} > \bar{P}_{Impala}$ e $Pr_{SD-Impala} < Pr_{Impala}$. Entretanto, a condição C2 foi atendida para os valores médio e máximo. Isso significa que, embora a técnica híbrida reduza os falso-negativos, ela aumenta os falso-positivos.

Profundidade = 10 e Distância de execução = 500						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{P}$	Pr	$\bar{P}$	Pr	$\bar{P}$	Pr
CollectEA	10	0,006	272,8	0,083	810	0,226
Impala	1	0,153	14,2	0,396	29	0,585
SD-Impala	35	0,061	100,2	0,151	203	0,285

Tabela 4.6: Falso-positivos e precisão para cenário (i)

Profundidade = 10 e Distância de execução = 500						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{P}$	Pr	$\bar{P}$	Pr	$\bar{P}$	Pr
CollectEA	10	0,006	272,8	0,083	810	0,226
Impala	1	0,341	19	0,341	41	0,506
SD-Impala	38	0,037	131,2	0,142	254	0,269

Tabela 4.7: Falso-positivos e precisão para cenário (ii)

Profundidade = 20 e Distância de execução = 1000						
Valores	**Mínimo**		**Média**		**Máximo**	
Ferramentas	$\bar{P}$	Pr	$\bar{P}$	Pr	$\bar{P}$	Pr
CollectEA	10	0,006	272,8	0,083	810	0,226
Impala	1	0,117	19	0,342	41	0,511
SD-Impala	38	0,031	153	0,134	319	0,269

Tabela 4.8: Falso-positivos e precisão para cenário (iii)

4.5 Discussão

4.5.1 Falso-negativos e Revocação

A Figura 4.2 representa a média dos valores dos falso-negativos do Impala, CollectEA e o SD-Impala em relação aos cenários (i), (ii) e (iii). O número de falso-negativos não se altera com a variação dos valores dos parâmetros no CollectEA. O número de falso-negativos do SD-Impala é menor do que o do CollectEA e o do Impala, ou seja, $\bar{N}_{Hibrida} < \bar{N}_{estatica}$ e $\bar{N}_{Hibrida} < \bar{N}_{dinamica}$. Em todos os cenários, o número de falso-negativos e a revocação do SD-Impala foram de acordo com nossas expectativas.

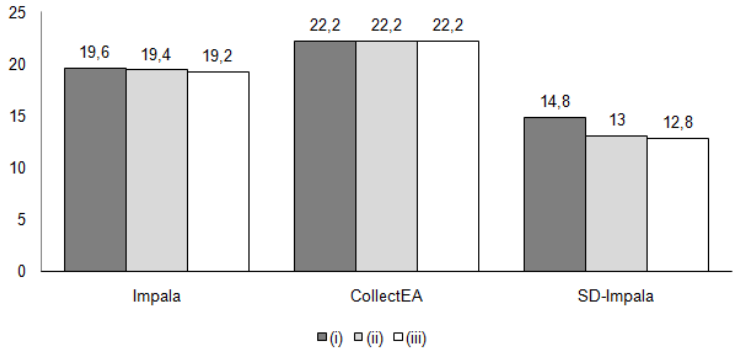


Figura 4.2: Número de falso-negativos *versus* ferramenta em cada um dos cenários

A Figura 4.3 representa a média dos valores da revocação do Impala, CollectEA e o SD-Impala em relação aos parâmetros (profundidade e distância). Para o CollectEA, a variação dos parâmetros não altera o valor da revocação. A revocação do SD-Impala é maior do que o do CollectEA e o do Impala, ou seja, $R_{Hibrida} > \bar{N}_{estatica}$ e $R_{Hibrida} > \bar{N}_{dinamica}$. Em média, a revocação do SD-Impala é sempre maior que a das demais ferramentas atendendo nossas expectativas.

Nos nossos experimentos, em média, a técnica híbrida aumentou a revocação e reduziu o número de falso-negativos.

O resultado dos experimentos também nos permite concluir que:

- SD-Impala aumentou a revocação entre 90% e 115% em relação ao Impala;
- SD-Impala aumentou a revocação entre 21,2% e 39% em relação ao CollectEA;

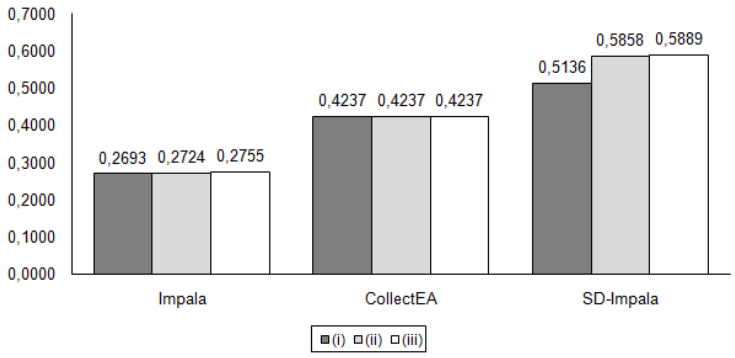


Figura 4.3: Valor da revocação *versus* ferramenta em cada um dos cenários

- aumentar a profundidade, aumentou a revocação do Impala em 2,3%;
- aumentar a profundidade e distância de execução, aumentou a revocação do SD-Impala em 14,6%;

4.5.2 Falso-positivos e Precisão

A Figura 4.4 apresenta a relação entre o número de falso-positivos das ferramentas em relação aos cenários (i), (ii) e (iii). O CollectEA obteve mais falso-positivos em relação ao SD-Impala e Impala. É importante observar que, as referências bibliográficas apontam a abordagem estática como àquela que apresenta maior número de falso-positivos. O número de falso-positivos devido a utilização de parâmetros tanto no SDImpala quanto no Impala.

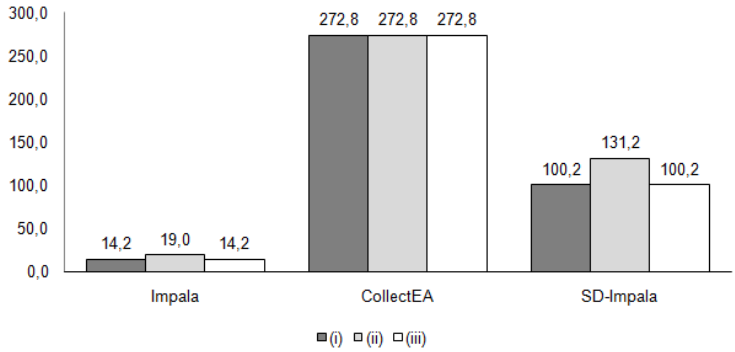


Figura 4.4: Números de falso-positivos *versus* ferramenta em cada um dos cenários

A Figura 4.5 apresenta a relação entre a precisão das ferramentas e os valores dos parâmetros. O CollectEA teve a menor precisão quando comparada às demais ferramentas

analisadas. A precisão do SD-Impala é menor do que o Impala. A medida que aumentamos a profundidade e a distância de execução, reduzimos a precisão. Isso representa que aumentar o valor dos parâmetros representou o aumento dos falso-positivos.

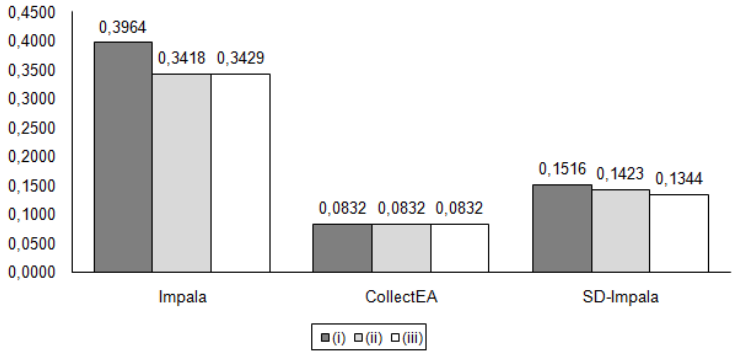


Figura 4.5: Valor da precisão *versus* ferramenta em cada um dos cenários

Como conclusão desses resultados, temos:

- $Pr_{SD-Impala} > Pr_{Impala}$ e $\bar{P}_{SD-Impala} > \bar{P}_{Impala}$
- $Pr_{SD-Impala} < Pr_{Dinamica}$ e $\bar{P}_{SD-Impala} < \bar{P}_{CollectEA}$
- SD-Impala reduziu a precisão entre 58,4% e 61% em relação ao Impala;
- SD-Impala aumentou a precisão entre 61,5% e 82% em relação ao CollectEA;
- aumentar a profundidade, aumentou a precisão do Impala em 13,4%;
- aumentar a profundidade e distância de execução, aumentou a precisão do SD-Impala em 11,3%;

4.6 Considerações Finais

Neste capítulo, apresentamos o estudo para avaliar a efetividade da técnica híbrida de análise de impacto. Descrevemos a metodologia utilizada para realização dos experimentos que consistiu em analisar o impacto de mudanças utilizando ferramentas diferentes e comparar seus resultados. As nossas conclusões sobre o resultados dos experimentos são:

- Em média, o SD-Impala obteve maior revocação e menor número de falso-negativos, se comparado com o Impala e o CollectEA;
- Em média, o SD-Impala obteve maior precisão e menor número de falso-positivos, se comparado ao CollectEA;
- Em média, o SD-Impala obteve menor precisão e maior número de falso-positivos, se comparado com o Impala;

Assim, embora produza menos falso-negativos do que as duas técnicas utilizadas, o resultado do SD-Impala possui mais falso-positivos do que o Impala. De acordo com o resultado dos experimentos, podemos concluir que a combinação de abordagem nos permite reduzir o número de falso-negativos, porém, produziu mais falso-positivos.

Nossa hipótese foi validada para o contexto dos nossos experimentos. Como nosso estudo foi realizado em projetos relativamente pequenos (ver Tabela 4.1), nossos resultados podem estar limitados. O tamanho dos projetos está diretamente relacionado a um problema encontrado na técnica que desenvolvemos: escalabilidade. Em projetos grandes, os rastros poderiam atingir mais de 60 GigaBytes de informação, inviabilizando o seu processamento devido a grande informação armazenada e gerada pelo SD-Impala.

Para melhorar a metodologia utilizada, seria necessário selecionar um número maior de projetos e ferramentas de análise de impacto e realizar uma análise estatística desses resultados, para termos uma confiança maior sobre a validação da hipótese. Mas devido ao tempo restrito do trabalho, optamos por sugerir esse tipo de estudo como trabalhos futuros.

Capítulo 5

Ferramenta: *SD*-Impala

Para avaliar a viabilidade da técnica proposta nesse trabalho, implementamos o protótipo de analisador de impacto ***S**tatic**D**ynamic*-Impala, ou simplesmente, *SD*-Impala. Desenvolvemos o SD-Impala no contexto do projeto Design Checker e utilizamos o seu processo de desenvolvimento que está no Apêndice B.

A nossa solução é específica para sistemas orientados a objetos, mas o SD-Impala é implementado para análise de impacto apenas para sistemas desenvolvidos em Java. Assim, consideramos as características próprias dessa linguagem para guiar na identificação de dependências estáticas e dinâmicas. Esse capítulo apresenta os 5 componentes do SD-Impala para realização da análise de impacto: Impala que identifica as dependências estruturais entre as entidades através da análise do código fonte do sistema, Event Collector que obtém os rastros de execução e o Dynamic Analyzer que os analisa para identificar dependências dinâmicas, Ranker e o Impala Viewer para combinação e ponderação dos resultados estáticos e dinâmicos.

5.1 Visão Geral

O SD-Impala é um protótipo desenvolvido em Java que implementa a técnica apresentada no Capítulo 3. Basicamente, a técnica híbrida é composta por 3 etapas:

1. Identificação de dependência estrutural entre as demais entidades do sistema e a mudança utilizando a análise estática;

2. Identificação de dependência dinâmica entre as entidades utilizando a análise de rastros de execução;

3. Combinação e ponderação dos resultados estático e dinâmico;

O nosso protótipo possui 5 componentes: o Impala é responsável pela etapa 1, Event Collector e Dynamic Analyzer pela etapa 2 e Ranker e o Impala Viewer pela 3. A Figura 5.1 representa a visão geral dos componentes do SD-Impala.

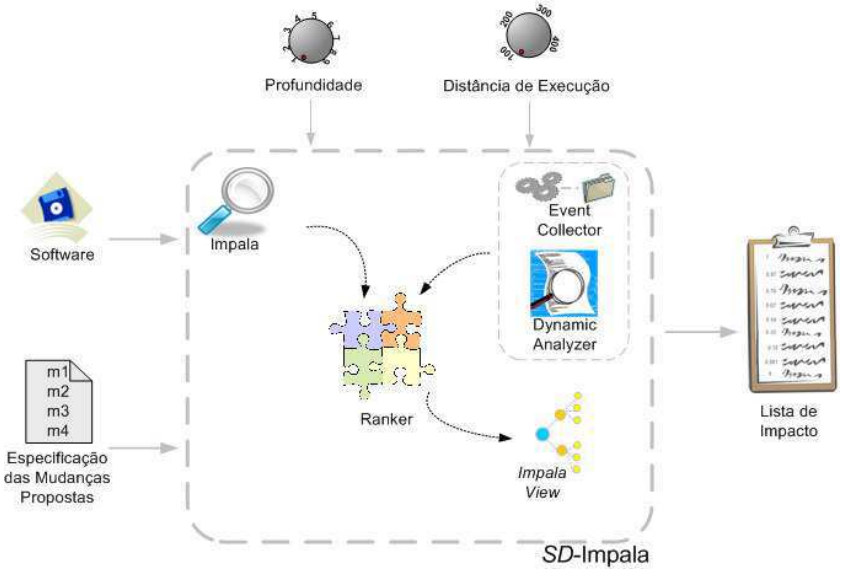


Figura 5.1: Visão Geral do SD-Impala

De acordo com a Figura 5.1, o usuário é responsável por definir as mudanças, ajustar os parâmetros e obter os artefatos do sistema que deve ser analisado. Essas informações são repassadas para o SD-Impala como parâmetros. As mudanças são definidas utilizando uma linguagem de *script* chamada *Change Description Language* (CDL). A Tabela 5.1 representa os comandos que podem ser descritos utilizando a CDL.

Os comandos relacionados a métodos possuem um campo optativo: parâmetros. Caso existam métodos com mesmo nome, os parâmetros devem ser passados no comando. Consideramos que o nomeDaClasse representa o identificador da classe que é formado pelo pacote seguido de "‘."’ e o nome da classe. Cada comando representa a proposta de uma mudança no sistema e deve ser passado como parâmetro para o SD-Impala em um arquivo textual. Para exemplificar, a Figura 5.2 apresenta mudanças propostas para o ToyExample

Operação	**Entidade**	**Comando**
Inserção	Classe	add <pacote>.<nomeDaClasse>
Inserção	Atributo	add <pacote>.<nomeDaClasse>.<nomeDoAtributo>
Inserção	Método	add <pacote>.<nomeDaClasse>.<nomeDoMétodo> [parâmetros]
Remoção	Classe	remove <pacote>.<nomeDaClasse>
Remoção	Atributo	remove <pacote>.<nomeDaClasse>.<nomeDoAtributo>
Remoção	Método	remove <pacote>.<nomeDaClasse>.<nomeDoMétodo> [parâmetros]
Alteração	Atributo	changeField <pacote>.<nomeDaClasse>.<nomeDoAtributo>
Alteração de assinatura	Método	changeSignature <pacote>.<nomeDaClasse>.<nomeDoMétodo> [parâmetros] to <novaAssinatura>
Alteração de semântica	Método	changeSemantic <pacote>.<nomeDaClasse>.<nomeDoMétodo> [parâmetros]
Inserir herança	Classe	addInheritance <pacote>.<subtipo> <pacote2>.<supertipo>
Remover herança	Classe	removeInheritance <pacote>.<subtipo> <pacote2>.<supertipo>

Tabela 5.1: Change Description Language

utilizando a CDL. A primeira linha representa a mudança da assinatura do método getLabel() para getLabel(int), remover addmenuItem(CommandWithLabel) e alterar a assinatura do método addMenuItem(String) para addMenuItem(int).

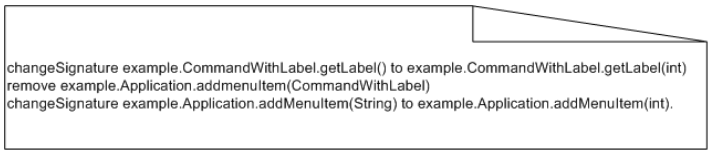


Figura 5.2: Exemplo de mudanças utilizando a CDL

Além do arquivo de mudanças, o SD-Impala precisa de mais 4 parâmetros de entrada: profundidade, código fonte, distância de execução e os rastros de execução. Os dois primeiros são utilizados para a análise estática e os últimos para dinâmica. O código fonte pode ser substituído pelos arquivos .jar[1] ou .war[2] contendo o *byte code* do sistema. Os rastros de execução são armazenados em arquivos de log e devem seguir uma sintaxe específica definida mais adiante.

O usuário interage com a interace do SD-Impala utilizando uma interface via linha de comando. Utilizando essa interface, o usuário executa o SD-Impala da seguinte forma:

java -jar sdImpala.jar <change file> <depth> <distance> <jar file> <logs directory>

Onde sdImpala.jar é o arquivo jar que representa o SD-Impala, <change file> é o arquivo de especificação de mudanças, depth representa o valor da profundidade e <distance> da distância de execução, <jar file> é o arquivo com o jar do sistema e <logs> o diretório onde estão os arquivos de log.

5.2 Análise Estática

O **Impala** é um protótipo de analisador de impacto que tem como foco sistemas orientados a objetos desenvolvidos em Java [7]. Utilizamos esse protótipo como o componente responsável por identificar a dependência estrutural entre as entidades do sistema e as mudanças propostas. A dependência estrutural é determinada pelo relacionamento entre as entidades

[1] *Java Archive*
[2] *Web ARchive file*

obtida através da análise do código fonte do sistema. O Impala representa o sistema em um grafo onde vértices e arestas representam as entidades e seus relacionamentos, respectivamente. O analisador realiza uma busca em profundidade para identificar as entidades que relacionam-se com as mudanças propostas. O usuário deve quão abrangente deve ser a análise através do parâmetro profundidade que representa o ponto de parada da busca realizada no grafo.

5.3 Análise Dinâmica

A segunda etapa consiste na análise dos rastros de execução para identificação de dependência dinâmica entre as demais entidades do programa e as entidades da mudança. O SD-Impala possui componentes para geração e análise dos dados de execução do sistema: Event Collector e Dynamic Analyzer. O primeiro é responsável por coletar e armazenar os rastros com os dados sobre a execução do programa. E o seguinte análise esses dados e gera o mapa de sucessores dos eventos da mudança.

5.3.1 *Event Collector*

O Event Collector é o componente do SD-Impala responsável pela coleta e armazenamento de informações sobre a execução do programa. O componente instrumenta o código fonte utilizando o AspectJ que é uma plataforma poderosa para desenvolvimento de sistemas em Java utilizando aspectos [12]. Ao executar o programa, o aspecto do Event Collector armazena dados sobre os eventos das entidades em arquivos de log. Esse log representa o rastro de execução do sistema com os eventos das entidades e a ordem em queforam executados. Como apresentamos na Seção 5.3.1, nos interessam apenas eventos de entrada e retorno de método e escrita e leitura de atributos. A Figura 5.3 representa um trecho de rastro de execução do ToyExample. A notação utilizada para representar um evento é <tipo de evento> + <identificador da entidade>, onde <tipo de evento> é o identificador do tipo.

De acordo do projeto, o tamanho do log com o rastro de execução pode tornar a análise lenta. Para o caso do Design Wizard, um dos projetos utilizados para avaliação do SDImpala, que possui 5825 linhas de código (LOC), o tamanho dos logs chegou a atingir 65Gb e seu processamento durou mais de 4 dias. Além do tempo, a análise desses logs consome

```
E example.command.MacroCommand.contains(framework.Command)
ER example.factory.Factory.createCommand(java.lang.String)
W example.command.MacroCommand.label
R example.command.MacroCommand.contains(framework.Command)
E example.command.MacroCommand.<init>()
L example.command.MacroCommand.label
R example.command.MacroCommand.<init>()
```

Figura 5.3: Exemplo de trecho do rastro de execução do ToyExample

memória para armazenamento do mapa de sucessores. Com logs suficientemente grandes (65G, por exemplo), obtivemos erros do tipo "'memória RAM insuficientes"' em máquinas Intel(R) Core(TM)2 CPU 6600 @ 2.40GHz, Memória 2GHz e HD 160 GB. Para reduzir o tamanho dos logs e da memória utilizada na geração do mapa de sucessores, o Event Collector utiliza um dicionário de dados que mapeia a entidade do evento a um identificador inteiro.

A Figura 5.4 representa o trecho de log da Figura 5.3 com a utilização de dicionário de dados. Durante a análise do byte code do projeto, geramos o dicionário de dados associando um valor inteiro como identificador de cada entidade. A Figura 5.5 representa um exemplo de dicionário de dados gerado para análise do ToyExample. A utilização do dicionário permite que o Event Collector armazena um rastro menor porque substitui o nome da entidade por identificador inteiro. Por exemplo, invés de referenciar *example.command.MacroCommand.contains(framework.Command)*, o log armazena o identificador correspondente: 18. Em algumas situações, a utilização do dicionário de dados reduziu o tamanho do log de 65Gb para 3Gb (redução de mais de 95%). Outro problema com o tamanho do log estava no tempo de processamento, a utilização do dicionário reduziu de 4 dias para 6 horas.

```
E 18
ER 128
W 22
R 18
E 19
L 22
R 19
```

Figura 5.4: Exemplo de log com a utilização de dicionário de dados

```
18 method#example.command.MacroCommand.contains(framework.Command)
19 method#example.command.MacroCommand.<init>()
22 field#example.command.MacroCommand.label
128 method#example.factory.Factory.createCommand(java.lang.String)
```

Figura 5.5: Exemplo de dicionário de dados

5.3.2 *Dynamic Analyzer*

O DynamicAnalyzer é responsável pela análise do rastro de execução gerado pelo EventCollector e armazenamento de informações sobre a dependência dinâmica entre as entidades. Dizemos que as entidades são dependentes dinamicamente se existe uma relação de sucessão entre seus eventos. Portanto, o principal objetivo do Dynamic Analyzer é identificar os sucessores dos eventos associados às entidades das mudanças e armazená-los no mapa de sucessão.

É um componente também desenvolvido em Java e implementa a técnica apresentada na Seção 3.3. O Dynamic Analyzer analisa o rastro de execução gerado pelo Event Collector e identifica os k-sucessores dos eventos associados às entidades das mudanças. Esses sucessores são armazenados em uma estrutura chamada de Mapa de Sucessores. O mapa de sucessores armazena os k-sucessores indexados pela entidade da mudança. O elemento do mapa contém uma tupla com três elementos (k, evento, ocorrência), onde k representa a distância entre o sucessor e o evento da entidade da mudança e a ocorrência é o número de vezes que o evento aparece como sucessor da mudança.

A Figura 5.6 representa como o mapa de sucessores para o rastro da Figura 5.4. Utilizamos um HashMap de Java para armazenar uma lista de sucessores associados às entidades da mudança. Cada elemento corresponde a tupla que representa o sucessor. Para exemplificar, vamos destacar a segunda linha do mapa onde são armazenados os sucessores dos eventos de Application.addMenuItem(CommandWithLabel). Nessa linha, a tripla (180,PasteCommand.<init>(Document)$_E$,1) representa que PasteCommand.<init>(Document)$_E$ foi uma vez 180-sucessor dos eventos de Application.addMenuItem(CommandWithLabel).

Key	List<Sucessor>
Application.addMenuItem(String)	(3, Document.<init>(String)$_R$,1), (9, Document.open()$_E$, 1), (14, Document.open()$_E$, 1), (62, Document.open()$_E$, 1), (11, Document.open()$_R$, 1), (16, Document.open()$_R$, 1), (64, Document.open()$_R$, 1)
Application.addMenuItem (CommandWithLabel)	(180, PasteCommand.<init>(Document)$_E$,1), (183, PasteCommand.<init>(Document)$_R$,1)

Figura 5.6: Exemplo de implementação do mapa de sucessores

5.4 Combinação e Ponderação do Resultado

A última etapa da análise de impacto consiste em combinar os resultados obtidos pelas etapas anteriores e classificá-los de acordo com as informações dinâmicas. Conforme foi apresentado na Seção 3.4, durante a análise dinâmica, armazenamos informações sobre o número de ocorrência dos sucessores dos eventos da mudança no mapa de sucessores. Essas informações são utilizadas durante essa fase para classificar as entidades e ordená-las utilizando um fator de impacto baseado na probabilidade de ocorrência dos eventos como sucessores.

5.4.1 Ranker e Impala Viewer

O Ranker é o componente responsável por unir e ponderar os resultados produzidos pelo Impala e o Dynamic Analyzer. Para ponderar o resultado, calculamos o fator de impacto baseando-se em informações produzidas pelo Dynamic Analyzer. Esse componente itera pelos sucessores dos eventos associados à mudança e pelas entidades obtidas pelo Impala e calcula o fator de impacto para cada uma dessas entidades.

Esse componente implementa o algoritmo do Código 3.1 e produz uma lista de entidades ordenadas de acordo com esse fator de impacto. Essa lista é apresentada de maneira mais visualmente organizada pelo Impala Viewer. Esse componente gera um relatório em HTML que facilita a visualização dos impactos. O Viewer permite que o resultado seja ordenado de acordo com a ordem crescente e decrescente do fator de impacto e na ordem alfabética das entidades.

5.5 Considerações Finais

Esse capítulo apresentou o SD-Impala que é um protótipo desenvolvido para avaliar a técnica híbrida descrita no Capítulo 3. A técnica híbrida realiza a análise de impacto em três etapas: (i) identificação de dependências estruturais entre as entidades do sistema; (ii) identificação de dependências dinâmicas; e (iii) combinação e classificação dos resultados encontrados nas etapas anteriores. No SD-Impala, a etapa (i) é realizada pelo Impala, o analisador estrutural do Impala. A etapa (ii) é realizada por dois componentes: Event Collector e o Dynamic Analyzer. O primeiro coleta informações durante a execução do programa e o segundo, gera o mapa de sucessão. A última etapa também é realizada por dois componentes: Ranker e o ImpalaViewer. O Ranker combina e classifica os resultados, enquanto o Viewer gera um relatório de impacto contendo informação mais organizada visualmente.

Capítulo 6

Trabalhos Relacionados

Com a necessidade de reduzir os custos associados à estimava errada dos efeitos de mudanças, a análise de impacto tem sido foco de muitas pesquisas. Esse capítulo apresenta trabalhos na área semelhantes e os compara com a nossa técnica híbrida.

O Impala é um protótipo desenvolvido no Grupo de Métodos Formais que inclui técnicas de inspeção de código fonte e análise de repositório para estimativa de impacto de mudanças [7]. Possui 3 componentes: Impala Static que analisa a estrutura do programa a fim de identificar dependências entre as entidades, Impala Plugin que coleta informações do repositório sobre a evolução do programa e as repassa para o componente Miner que classifica o impacto obtido pelo Static de acordo com as informações coletadas pelo Plugin. Assim como na nossa solução, o resultado dessa técnica é uma lista de entidades impactadas. Esse trabalho assemelha-se com o nosso porque utiliza novas informações para incrementar a análise estrutural, além de produzir um resultado em forma de lista ordenada por impactos. Entretanto, sua aplicação em contextos com utilização de técnicas como very late binding pode gerar muitos falso-negativos.

Michele Lee é especialista em análise de impacto aplicada sistemas orientados a objetos. Elaborou uma técnica estática que representa o relacionamento entre as entidades do sistema em um grafo de dependência de dados. A partir desse grafo, ocorre a análise da dependência entre as entidades e os impactos são identificados. Esse trabalho assemelha-se com a técnica utilizada pelo Impala para estimativa de impacto. A diferença entre as duas técnicas está no fato do Impala considerar o parâmetro profundidade para como critério de parada. A técnica utiliza métricas conhecidas como “métricas de impacto de mudanças orientadas a

objetos" para classificar os impactos. Assim, de acordo com essas métricas, as entidades são classificadas como contaminada, limpa, semi-contaminada ou semi-limpa [15; 14]. E podem ser utilizadas para avaliar qual a melhor alternativa de mudança deve ser realizada. No nosso trabalho, utilizamos métricas para mensurar o erro relacionado à imprecisão da técnica. Esse trabalho assemelha-se com o nosso devido a técnica utilizada para estimar o impacto em sistemas orientados a objetos. Assim como em outras técnicas estática, produz falso-negativos em contextos com very late binding.

Barbara Ryder e colaboradores utilizam a abordagem dinâmica para análise de impacto. A análise é realizada em duas versões do programa P e P', onde P' representa o programa P após a implementação da mudança M. Assim, uma mesma suíte de testes é executada para P e P'. Os resultados dos testes nas duas versões são analisados e, o impacto da mudança M é identificado [24]. Embora classicamente a análise dinâmica seja aplicada para análise de impacto posterior à implementação da mudança, na técnica híbrida, a análise é realizada a priori. Além da necessidade da implementação da mudança para ser realizada a análise de impacto, a técnica de Ryder se difere da nossa porque não inclui classificação de resultados.

Lulu Huang e colaboradores desenvolveram uma técnica de análise dinâmica de impacto aplicada a sistemas orientados a objetos. Nesse trabalho, o impacto é definido pela análise de rastros de execução para identificação da dependência entre os métodos. Além disso, a técnica define algoritmos aplicados às mudanças em nível de atributo, ainda não consideradas em outras técnicas conhecidas de análise dinâmica. Entretanto, essa abordagem requer que a mudança tenha sido implementada, dificultando a utilização em atividades de planejamento [10]. Por esse motivo, esse trabalho se difere da técnica híbrida elaborada no nosso trabalho.

Entretanto, nenhuma das técnicas acima citadas possuem uma abordagem híbrida. O Coverage Impact é considerado híbrido porque a análise de impacto é realizada em duas etapas: análise dinâmica e estática. Na primeira, informações dinâmicas são coletadas e um vetor de bits é gerado. Esse vetor permite concluir quais os métodos que aparecem nos rastros em cada uma das execuções do programa. A segunda etapa da análise consiste em calcular o corte de encaminhamento estático [1]. O resultado do Coverage Impact é um conjunto contendo as entidades consideradas impactadas pelas duas etapas [18]. Entretanto, essa técnica é considerada imprecisa quando comparada com técnicas dinâmicas porque não

[1] Do inglês *static forward slice*

considera a ordem em que os eventos aparecem no rastro [2]. Essa técnica se difere da nossa em duas questões: 1) análise de impacto após à mudança ser realizada e 2) análise dinâmica antes da estática. A primeira questão, inviabilizou a utilização do Coverage Impact como técnica de controle durante nossos estudos na avaliação da hipótese. Acreditamos que a segunda questão pode representar a diferença na precisão dessa técnica em relação às outras investigadas em [2].

Capítulo 7

Considerações Finais

As técnicas de análise de impacto são imprecisas porque apresentam erros, ou seja, podem incluir entidades não afetadas ou desconsiderar entidades afetadas [18; 9; 10]. Quando o resultado da análise possui entidades não afetadas, dizemos que este apresenta falso-positivos. Por outro lado, Quando o resultado desconsidera entidades afetadas, dizemos que este apresenta falso-negativos. O primeiro erro caracteriza a superestimativa do impacto, com a presença de informações irrelevantes para análise. O segundo caracteriza a subestimativa, quando a análise oculta impactos. Ambas as situações causam prejuízo à empresa.

Esse trabalho teve como foco o problema dos falso-negativos encontrados pelas técnicas que utilizam as abordagens clássicas separadamente. Para isso, propusemos uma técnica que combina abordagens estática e dinâmica para realização da análise de impacto. Dessa forma, validamos nosso trabalho de duas formas: 1) a viabilidade e 2) a efetividade da proposta de combinar as abordagens estática e dinâmica. A viabilidade foi demonstrada pela implementação da ferramenta conhecida como SD-Impala. A efetividade, por sua vez, foi avaliada através do estudo que comparou resultados de ferramentas de análise de impacto diferentes.

A técnica foi dividida em 3 etapas: análise estática, análise dinâmica e combinação de resultados. Na primeira etapa, identificamos relações de dependências entre as entidades analisando a estrutura do código fonte. Na segunda, identificamos dependências dinâmicas através da análise de rastros de execução do programa. Na última etapa, os resultados das etapas anteriores são combinados e classificados de acordo com o fator de impacto. Esse fator de impacto é determinado pela relação de dependência entre as entidades.

A validação da técnica proposta foi realizada em duas etapas: 1) implementação de um protótipo e 2) estudo de comparação de resultados. Na primeira, avaliamos viabilidade da técnica desenvolvendo um protótipo de analisador de impacto chamado de SD-Impala. O protótipo é composto por 5 componentes: Impala, Event Collector, Dynamic Analyzer, Ranquer e Impala Viewer. O Impala é um analisador de impacto estático que tem como objetivo estimar o impacto em sistemas orientados a objetos através da análise do código fonte do programa. O Event Collector é responsável por coletar os rastros de execução. O Dynamic Analyzer identifica as relações de dependência dinâmica entre as entidades a partir do rastro gerado pelo Event Collector. O Ranquer combina e classifica os resultados obtidos pelo Impala e pelo Dynamic Analyzer. Por fim, o Impala Viewer gera relatórios de impacto em um formato mais organizado visualmente.

Na segunda etapa, realizamos um estudo para comparar o resultado de ferramentas de análise de impacto que utilizam abordagens diferentes. Dessa forma, selecionamos projetos de software em andamento, realizamos a análise de impacto utilizando o Impala, CollectEA e o SD-Impala e comparamos os erros encontrados em seus resultados. Para mensurar os erros dos resultados, utilizamos as métricas falso-negativos, falso-positivos, revocação e precisão. Consideramos que revocação é inversamente proporcional ao número de falso-negativos e precisão inversamente proporcional aos falso-positivos. Para que a hipótese fosse considerada válida nos nossos experimentos, seria necessário que as seguintes condições fossem atendidas:

1. $\bar{N}_{SD-Impala} < \bar{N}_{Impala}$ e $\bar{N}_{SD-Impala} < \bar{N}_{CollectEA}$;
2. $R_{hibrida} > R_{estatica}$ e $R_{hibrida} > R_{dinamica}$

Nos nosso estudo, o SD-Impala aumentou a revocação entre 90% e 115% em relação ao Impala, e aumentou entre 21,2% e 39% em relação ao CollectEA. Esses resultados representam que o SD-Impala obteve menos falso-negativos para ambos os casos. Em nosso estudo, a hipótese investigada estava válida e justifica a utilização da técnica híbrida para análise de impacto.

7.1 Contribuições

O principal mérito desse trabalho é uma técnica híbrida de análise de impacto que pode ser aplicada antes da realização da mudança e que produz menos falso-negativos do que outras técnicas que utilizam a abordagem tradicional. Com a combinação de técnicas que utilizam abordagens diferentes, conseguimos obter valores de revocação maiores do que a técnica estática e a dinâmica. A nossa solução reduziu a precisão dos resultados e, consequentemente, aumentou o número de falso-positivos, em relação à técnica estática. Entretanto, o ganho da revocação, quando comparamos as técnicas híbrida e estática, foi maior.

Outra contribuição da nossa técnica foi a forma de apresentação dos resultados. Na maioria das técnicas de análise de impacto, o resultado é representado por um conjunto contendo as entidades consideradas afetadas pela mudança. No nosso trabalho, o resultado é uma lista ordenada de acordo com a dependência dinâmica entre as entidades. Essa forma de apresentação do impacto auxilia o analista na análise com a disponibilidade de informação mais organizada.

A técnica híbrida de análise de impacto estática possui algumas limitações, como o aumento do número de falso-positivos e, consequentemente, redução da precisão. Em contextos onde a presença de falso-positivos deve ser limitada, não devemos utilizar a técnica híbrida. Entretanto, utilizar a abordagem híbrida nos permite reduzir o erro associado à subestimativa de impacto. Em alguns contextos, como em fábricas de software, superestimar não é tão crítico quanto subestimar. Por esse motivo, acreditamos que a técnica híbrida pode ser uma abordagem interessante para auxiliar analista no planejamento de custos das mudanças.

Outra limitação da técnica híbrida está no fato dos seus resultados serem dependentes da cobertura dos testes utilizados para coleta da informação dinâmica. Essa limitação também atinge as demais técnicas dinâmicas. Isso acontece porque, para obter dados relevantes para análise, a execução do programa deve atingir o máximo possível do código. Portanto, em situações onde não há testes ou são insuficientes, talvez não seja uma boa ferramenta utilizar a abordagem híbrida. Em contextos como esse, o resultado da técnica híbrida se assemelha à estática.

Além disso, contribuímos com o protótipo de analisador de impacto SD-Impala que pode

ser para o planejamento dos custos da mudança. Entretanto, sua utilização em sistemas significantemente grandes é limitado devido à quantidade de informação gerada e armazenada pelo componente Dynamic Analyzer.

7.2 Trabalhos futuros

Com a conclusão deste trabalho, encontramos algumas sugestões para complementá-lo. Essas sugestões de melhorias ou trabalhos futuros foram listamos abaixo.

A solução desenvolvida para realizar a análise dinâmica apresentada na Seção 3.3 apresenta problemas de desempenho e escalabilidade. Isso acontece devido a grande quantidade de dados gerada e armazenada no mapa de sucessores. Como trabalhos futuros, propomos a **evolução da técnica dinâmica** de modo a permitir que a nossa solução possa ser facilmente estendida e que dê suporte a análise de impacto em sistemas de grande porte. Nesse contexto, há possibilidade de melhorar a técnica desde o formato do rastro de execução gerado pelo coletor de eventos até a estrutura de armazenamento dos sucessores.

O módulo de análise estática da solução híbrida analisa o código fonte a fim de identificar as dependências estruturais entre as entidades do sistema. Entretanto, a análise estática pode levar em consideração outros artefatos ou também outras características do programa. Por esse motivo, propomos a **utilização de outras técnicas de análise estática** para aumentar a precisão dos resultado da técnica híbrida.

Realizamos um estudo para verificar a efetividade de nossa técnica. Esse estudo verificou variáveis quantitativas sobre o resultado da análise de impacto. Propomos a realização de uma **pesquisa de campo** para verificar variáveis qualitativas, como por exemplo, a satisfação do analista.

Bibliografia

[1] Taweesup Apiwattanapong. *Identifying testing requirements for modified software*. PhD thesis, Georgia Institute of Technology, 2007.

[2] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Efficient and precise dynamic impact analysis using execute-after sequences. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 432–441, New York, NY, USA, 2005. ACM.

[3] B. Breech, A. Danalis, S. Shindo, and L. Pollock. Online impact analysis via dynamic compilation technology. pages 453–457, Sept. 2004.

[4] L. C. Briand, Y. Labiche, and L. O'Sullivan. Impact analysis and change management of uml models. In *ICSM '03: Proceedings of the International Conference on Software Maintenance*, page 256, Washington, DC, USA, 2003. IEEE Computer Society.

[5] João Brunet. Design wizard. http://www.designwizard.org/, 2009.

[6] Len Erlikh. Leveraging legacy system dollars for e-business. *IT Professional*, 2(3):17–23, 2000.

[7] Lile Palma Hattori. Análise probabilística de impacto de mudanças baseada em históricos de mudanças de software. Master's thesis, Universidade Federal de Campina Grande, Campina Grande, Brasil, Fevereiro 2008.

[8] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. In *PLDI '88: Proceedings of the ACM SIGPLAN 1988 conference on Programming Language design and Implementation*, pages 35–46, New York, NY, USA, 1988. ACM.

[9] Lulu Huang and Yeong-Tae Song. Dynamic impact analysis using execution profile tracing. *sera*, 0:237–244, 2006.

[10] Lulu Huang and Yeong-Tae Song. Precise dynamic impact analysis with dependency analysis for object-oriented programs. In *SERA '07: Proceedings of the 5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*, pages 374–384, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Suhaimi Ibrahim, Norbik Idris, Malcolm Munro, and Aziz Deraman. A software traceability validation for change impact analysis of object oriented software. In Hamid R. Arabnia and Hassan Reza, editors, *Software Engineering Research and Practice*, pages 453–459. CSREA Press, 2006.

[12] Ramnivas Laddad. *AspectJ in Action: Practical Aspect-Oriented Programming*. Manning Publications Co., Greenwich, CT, USA, 2003.

[13] James Law and Gregg Rothermel. Whole program path-based dynamic impact analysis. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 308–318, Washington, DC, USA, 2003. IEEE Computer Society.

[14] Michelle Lee, A. Jefferson Offutt, and Roger T. Alexander. Algorithmic analysis of the impacts of changes to object-oriented software. In *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, page 61, Washington, DC, USA, 2000. IEEE Computer Society.

[15] Michelle L. Lee. Change impact analysis of object-oriented software. Master's thesis, George Mason University,, 1995.

[16] Mikael Lindvall. An empirical study of requirements-driven impact analysis in object-oriented software evolution. In *Linköping University, Institute of Technology, Sweden*, 1997.

[17] Michael Ernst Mit and Michael D. Ernst. Static and dynamic analysis: Synergy and duality. In *In WODA 2003: ICSE Workshop on Dynamic Analysis*, pages 24–27, 2003.

[18] Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold. 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*, pages 128–137, New York, NY, USA, 2003. ACM.

[19] Alessandro Orso, Taweesup Apiwattanapong, James Law, Gregg Rothermel, and Mary Jean Harrold. An empirical comparison of dynamic impact analysis algorithms. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 491–500, Washington, DC, USA, 2004. IEEE Computer Society.

[20] Shari Lawrence Pfleeger and Shawn A. Bohner. A framework for software maintenance metrics. In *IEEE Transactions on Software Engineering*, pages 320–327, May 1990.

[21] Turver Richard J and Munro Malcolm. An early impact analysis technique for software maintenance. *The Journal of Software Maintenance, Research and Practice*, 1994.

[22] Shawn Bohner Robert Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[23] Gregg Rothermel and Mary Jean Harrold. Selecting tests and identifying test coverage requirements for modified software. In *International Symposium on Software Testing and Analysis*, pages 169–184, 1994.

[24] Barbara G. Ryder and Frank Tip. Change impact analysis for object-oriented programs. In *PASTE '01: Proceedings of the 2001 ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 46–53, New York, NY, USA, 2001. ACM.

Apêndice A

Resultados dos Experimentos

|I∩A|									
Dinâmica	6								
Estática	Prof=5			Prof=10			Prof=20		
	41			42			43		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	44	45	47	45	46	47	45	46	47

|A|	65

|I|								
Dinâmica	875							
Estática	Prof=5			Prof=10			Prof=20	
	70			83			84	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	247	250	269	291	300	304	353	360	366

Precisão								
Dinâmica	0,006857143							
Estática	Prof=5			Prof=10			Prof=20	
	0,585714286			0,506024096			0,511904762	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,178138	0,18	0,174721	0,154639	0,153333	0,154605	0,127479	0,127778	0,128415

Revocação								
Dinâmica	0,092307692							
Estática	Prof=5			Prof=10			Prof=20	
	0,630769231			0,646153846			0,661538462	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,676923	0,692308	0,723077	0,692308	0,707692	0,723077	0,692308	0,707692	0,723077

Falso-Positivos								
Dinâmica	810							
Estática	Prof=5			Prof=10			Prof=20	
	29			41			41	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	203	205	222	246	254	257	308	314	319

Falso-Negativos								
Dinâmica	59							
Estática	Prof=5			Prof=10			Prof=20	
	24			23			22	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	21	20	18	20	19	18	20	19	18

Figura A.1: Dados coletados durante os experimentos no Impala

|I∩A|									
Dinâmica	13								
Estática	Prof=5			Prof=10			Prof=20		
	2			2			2		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	10	10	10	10	10	10	10	10	10

|A|	18

|I|								
Dinâmica	215							
Estática	Prof=5			Prof=10			Prof=20	
	13			17			17	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	147	151	151	166	170	170	182	186	186

Precisão								
Dinâmica	0,060465116							
Estática	Prof=5			Prof=10			Prof=20	
	0,153846154			0,117647059			0,117647059	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,068027	0,066225	0,066225	0,060241	0,058824	0,058824	0,054945	0,053763	0,053763

Revocação								
Dinâmica	0,722222222							
Estática	Prof=5			Prof=10			Prof=20	
	0,111111111			0,111111111			0,111111111	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,555556	0,555556	0,555556	0,555556	0,555556	0,555556	0,555556	0,555556	0,555556

Falso-Positivos								
Dinâmica	197							
Estática	Prof=5			Prof=10			Prof=20	
	11			15			15	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	141	141	141	156	160	160	172	176	176

Falso-Negativos								
Dinâmica	5							
Estática	Prof=5			Prof=10			Prof=20	
	16			16			16	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	8	8	8	8	8	8	8	8	8

Figura A.2: Dados coletados durante os experimentos no Abstractor

|I ∩ A|									
Dinâmica	19								
Estática	Prof=5			Prof=10			Prof=20		
	5			5			5		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	12	12	12	18	18	18	18	18	18

|A|	32

|I|								
Dinâmica	84							
Estática	Prof=5			Prof=10			Prof=20	
	26			26			26	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	73	73	73	95	95	95	95	95	95

Precisão								
Dinâmica	0,226190476							
Estática	Prof=5			Prof=10			Prof=20	
	0,192307692			0,192307692			0,192307692	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,164384	0,164384	0,164384	0,18947	0,18947368	0,189474	0,189474	0,189474	0,189474

Revocação								
Dinâmica	0,59375							
Estática	Prof=5			Prof=10			Prof=20	
	0,15625			0,15625			0,15625	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,375	0,375	0,375	0,5625	0,5625	0,5625	0,5625	0,5625	0,5625

Falso-Positivos								
Dinâmica	65							
Estática	Prof=5			Prof=10			Prof=20	
	21			21			21	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	61	61	61	77	77	77	77	77	77

Falso-Negativos								
Dinâmica	13							
Estática	Prof=5			Prof=10			Prof=20	
	27			27			27	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	20	20	20	14	14	14	14	14	14

Figura A.3: Dados coletados durante os experimentos no ToyExample

|I ∩ A|									
Dinâmica	4								
Estática	Prof=5			Prof=10			Prof=20		
	1			1			1		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	4	4	4	5	5	5	5	5	5

|A|	7

|I|								
Dinâmica	289							
Estática	Prof=5			Prof=10			Prof=20	
	2			2			2	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	65	65	65	132	132	132	160	160	160

Precisão								
Dinâmica	0,01384083							
Estática	Prof=5			Prof=10			Prof=20	
	0,5			0,5			0,5	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,061538	0,061538	0,061538	0,037879	0,037879	0,037879	0,03125	0,03125	0,03125

Revocação								
Dinâmica	0,571428571							
Estática	Prof=5			Prof=10			Prof=20	
	0,142857143			0,142857143			0,142857143	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,571429	0,571429	0,571429	0,714286	0,714286	0,714286	0,714286	0,714286	0,714286

Falso-Positivos								
Dinâmica	282							
Estática	Prof=5			Prof=10			Prof=20	
	1			1			1	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	61	61	61	127	127	127	155	155	155

Falso-Negativos								
Dinâmica	3							
Estática	Prof=5			Prof=10			Prof=20	
	6			6			6	
Híbrida	Dist=100			dist=500			dist=1000	
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	3	3	3	2	2	2	2	2	2

Figura A.4: Dados coletados durante os experimentos no Design Wizard

|I ∩ A|									
Dinâmica	5								
Estática	Prof=5			Prof=10			Prof=20		
	11			11			11		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	14	14	14	14	14	14	14	14	14

|A|	36

Quantos a técnica encontrou?									
Dinâmica	46								
Estática	Prof=5			Prof=10			Prof=20		
	20			28			28		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	49	49	49	52	52	52	52	52	52

Precisão									
Dinâmica	0,108695652								
Estática	Prof=5			Prof=10			Prof=20		
	0,55			0,392857143			0,392857143		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,285714	0,285714	0,285714	0,269231	0,269231	0,269231	0,269231	0,269231	0,269231

Revocação									
Dinâmica	0,138888889								
Estática	Prof=5			Prof=10			Prof=20		
	0,305555556			0,305555556			0,305555556		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	0,388889	0,388889	0,388889	0,388889	0,388889	0,388889	0,388889	0,388889	0,388889

Falso-Positivos									
Dinâmica	10								
Estática	Prof=5			Prof=10			Prof=20		
	9			17			17		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	35	35	35	38	38	38	38	38	38

Falso-Negativos									
Dinâmica	31								
Estática	Prof=5			Prof=10			Prof=20		
	25			25			25		
Híbrida	Dist=100			dist=500			dist=1000		
	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20	Prof=5	prof=10	prof=20
	22	22	22	22	22	22	22	22	22

Figura A.5: Dados coletados durante os experimentos no Design

Apêndice B

Processo de Desenvolvimento de Software

A ferramenta foi desenvolvida no Laboratório de Métodos Formais da Universidade Federal de Campina Grande no contexto do projeto de pesquisa Design Checker que é uma parceria entre a universidade e a empresa CPMBraxis. Dessa forma, adotamos o processo de desenvolvimento de software desse projeto para guiar a implementação do *SD*-Impala.

A equipe do DesignChecker tem com base as práticas do eXtreme Programming adaptadas para o ambiente universitário, onde as pessoas envolvidas não estão presentes 40 horas por semana. As práticas adotadas pelo grupo são:

- Ambiente informativo - O ambiente de trabalho deve fornecer à equipe informações sobre o andamento do trabalho. Por esse motivo, utilizamos o programa XPlanner onde são visualizadas as User Stories e seus responsáveis e qual o tempo de realização previsto para cada atividade. O XPlanner é disponibilizado via Internet facilitando o acesso;

- Iteração - O ciclo adotado pela equipe tem a duração de 2 semanas. A cada iteração as User Stories são atualizadas em reuniões presenciadas com todos os membros da equipe;

- Reuniões de acompanhamento: Semanalmente, são realizadas reuniões para acompanhar o desempenho dos integrantes da equipe;

- Desenvolvimento orientado a testes - A equipe utiliza o JUnit como ferramenta para gerar testes durante a codificação do sistema;

- Programação em par - Essa prática garante que a equipe está em constante comunicação e garante que o conhecimento sobre o código está sempre sendo compartilhado;

As responsabilidades atribuídas a cada papel constante na equipe são:

- Coordenador: responsável por definir, especificar e validar o projeto a ser desenvolvido como um todo. Como também, gerenciar as políticas executivas de trabalho interno da equipe e da relação com o cliente;
- Gerente Financeiro: responsável pelo controle financeiro de todas as movimentações do projeto;
- Gerente de Projeto: responsável em integrar todos os membros da equipe e o cliente, facilitar a comunicação, controlar o progresso do trabalho e a qualidade do produto a ser desenvolvido, como também os riscos de projeto. Além de manter contato constante com o cliente, controlar e manter atualizado as informações referentes ao projeto, controlar o trabalho dos membros da equipe por horas de trabalho em cada tarefa, manter a coesão entre as linhas do projeto e fornecer um meio de controle de qualidade do processo de desenvolvimento como também do produto final, fazer o planejamento de releases (cronograma de liberação e instalação de um sistema usável e testável);
- Líder de Linha: responsável por definir, pesquisar e gerenciar os produtos a serem desenvolvidos de acordo com os requisitos fornecidos pelo cliente. Fornecendo além do software final, uma pesquisa sobre os temas relacionados e relevantes ao produto. Além de definir as UserSories de cada iteração referente a seu produto, procurar soluções especificas a cada tarefa de acordo com as pesquisas realizadas;
- Desenvolvedor: responsável por implementar as tarefas definidas pelas User Stories, definir as tarefas a serem realizadas em cada User Story, determinando tempo estimado, executar atividades de integração, executar testes de unidade e teste de aceitação (enviados pelo cliente) e refatoramento do código;

O processo utilizado pelo DesingChecker ocorre da seguinte forma:

1. Planejamento de Iteração

- Cada iteração possui o tempo de 2 semanas;
- Cada iteração abrange o Planejamento Gerencial, o Desenvolvimento e o planejamento e execução de testes;
- Para cada iteração deverá ser definida a ordem a ser desenvolvida as User Stories referenciadas pelo cliente, de acordo com sua importância e dependência entre elas;
- Para cada User Story é definida um grupo de tarefas, onde cada desenvolvedor alocará tempo (em horas, até preencher o total de horas semanais) para cada tarefa especificada;

2. Planejamento Gerencial. De acordo com as iterações definidas, serão realilzadas reuniões como segue:
   - Reunião mensal para troca de informações com o cliente sobre o andamento do projeto e as User Stories a serem desenvolvidas e alteradas;
   - Reunião quinzenal para definição ou alteração de User Story de cada produto, como também a discussão de soluções para problemas gerados durante a iteração;
   - Reunião semanal para discutir sobre o andamento da iteração e soluções de problemas de cada linha específica;
3. Desenvolvimento que corresponde a geração ou atualização de código;
4. Testes de aceitação e de unidade para reduzir a quantidade de erros no software produzido;