Raquel Fialho de Queiroz Lafetá

Uma Abordagem Híbrida para Construção de Documentação para Apoio à Instanciação de Frameworks

Tese de doutorado apresentada ao Programa de Pós-graduação da Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Marcelo de Almeida Maia

Uberlândia
2017

Uma Abordagem Híbrida para Construção de Documentação para Apoio à Instanciação de Frameworks

Raquel Fialho de Queiroz Lafetá

Universidade Federal de Uberlândia
Faculdade de Computação
Programa de Pós-Graduação em Ciência da Computação

Uberlândia
2017

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

L162a
2017

Lafetá, Raquel Fialho de Queiroz, 1983

Uma abordagem híbrida para construção de documentação para apoio à instanciação de frameworks / Raquel Fialho de Queiroz Lafetá. - 2017.

150 f.

Orientador: Marcelo de Almeida Maia.

Tese (doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Ciência da Computação.

Inclui bibliografia.

1. Computação - Teses. 2. Pórticos estruturais - Teses. 3. Programação (Computadores) - Métodos de ensino - Teses. I. Maia, Marcelo de Almeida. II. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU: 681.3

Aos meus pais José Antônio de Queiroz Lafetá e Regina Coeli Fialho
e meus irmãos Thiago e José Antônio Jr..
Ao meu esposo Tiago Leonel.
Ao Prof. Marcelo Maia.

Agradecimentos

Agradeço...

A Deus em primeiro lugar, por tudo que tem proporcionado em minha vida e por ser a maior herança que tenho. Minha fortaleza nos momentos mais difíceis. Tenho a certeza de que não ando só.

A meus pais José Antônio e Regina e meus irmão Thiago e Junior pelo amor, carinho e apoio nesta caminhada. Por ser a grande base de minha vida, pessoas que sei que sempre poderei contar e que me amam incondicionalmente. Ao meu irmão Thiago, agradeço por ter sido meu companheiro e amigo que me ajudou neste trabalho.

Ao meus esposo Tiago Leonel, por me ensinar a dar valor ao que realmente vale a pena nesta vida, os momentos que passamos com quem amamos.

Aos meus amigos e colegas da pós graduação por terem não só me apoiado, mas ajudado efetivamente nos resultados aqui apresentados. Sabia que podia contar com vocês, não imaginei que teria tanto apoio. Um agradecimento especial à minha amiga Lígia Passos por ter me acompanhado e apoiado nesta jornada.

Ao professor David Röthlisberger agradecemos pelas contribuições dadas à este trabalho, auxiliando na definição da abordagem. Obrigada.

Com grande gratidão agradeço ao professor Marcelo Maia pela paciência, apoio, dedicação e orientação, pelo grande profissional que é. Sentirei falta de nossas reuniões e conversas que tanto contribuíram para o meu aprendizado durante quase 10 anos de orientação. Obrigada.

"Por vezes sentimos que aquilo que fazemos não é senão uma gota de água no mar.
Mas o mar seria menor se lhe faltasse uma gota"
(Madre Teresa de Calcuta)

Resumo

Reuso de software é um dos principais objetivos em Engenharia de Software. *Frameworks* de aplicação promovem a reutilização de blocos de construção, mas também da solução arquitetural para um determinado domínio de aplicação. A criação de uma aplicação reutilizando um *framework* denomina-se instanciação do *framework* e requer um esforço substancial de compreensão do mesmo. Uma documentação de alta qualidade pode ser um instrumento útil para minimizar esse esforço. No entanto, na maioria dos casos, a documentação adequada não existe ou não é atualizada. Uma hipótese é que o próprio código fonte do *framework* e de instâncias existentes poderiam oferecer informação útil para novas instanciações. Contudo, haveria o desafio dos desenvolvedores entenderem quantidade substancial de código fonte. Neste contexto, o objetivo desta tese é demonstrar a viabilidade de construção de documentação relevante para a instanciação de *frameworks* utilizando análise estática e dinâmica do código fonte do *framework* e de suas instanciações pré-existentes. A proposta é apresentar tal documentação como um livro de receitas, onde as receitas são compostas de tarefas de programação e informações sobre os elementos do *framework* associados a uma característica de interesse. Inicialmente, dois estudos preliminares foram realizados para avaliar a cobertura e a utilidade prática das informações contidas nas receitas, os quais mostraram a necessidade de alguns ajustes, mas também indicaram receitas com informações relevantes e cobertura adequada. Por fim, foi conduzido um estudo robusto composto de 3 experimentos envolvendo ao todo 44 sujeitos humanos, com 88 execuções de atividades reais de instanciação de *frameworks*, onde o uso de livros de receitas foi comparado ao uso de documentações tradicionais dos *frameworks*. Os livros de receitas gerados semi-automaticamente apresentaram resultados de uso iguais ou melhores, em termos de taxa de acerto, tempo de execução e percepção da satisfação dos usuários, cumprindo os objetivos de pesquisa.

Palavras-chave: Compreensão de software. Engenharia reversa. Reuso. Framework. Livro de receitas. Análise estática. Análise dinâmica. Experimento controlado..

Abstract

Software reuse is one of the major goals in Software Engineering. Frameworks promote the reuse of individual building blocks, but also of system design. Framework instantiation is the construction of an application reusing a framework. This process requires a substantial understanding effort of the framework. So, high quality documentation may be a useful resource to minimize this effort. However, in most cases, appropriate documentation neither exists nor is up-to-date. A hypothesis is that the framework code itself and existing instantiations could provide useful information for new instantiations. However, in this case developers still would have to read large portions of code. The goal of this thesis is to demonstrate the feasibility of constructing relevant documentation for framework instantiation with static and dynamic analysis of the framework itself and pre-existing instantiations. The proposal is presenting the documentation in a cookbook style, where recipes are composed of programming tasks and information about framework elements related to a desired feature. Initially, two preliminary experiments were conducted to evaluate coverage and practical usefulness of the recipe information for developers. Results pointed out the need for some adjustments, but also indicated sufficient and relevant information in recipes. Finally, we performed a robust study, consisting of three experiments with 44 human subjects, and 88 executions of real framework instantiations. We compared the use of cookbooks with the use of traditional framework documentation. The generated cookbooks presented results better or as good as traditional framework documentation, in terms of correctness, time spent and the satisfaction perception of document uses.

Keywords: Program Comprehension. Reverse engineering. Reuse. Frameworks. Cookbook. Recipe. Static analysis. Dynamic analysis. Controlled experiment..

Lista de ilustrações

Figura 1 – Representação de uma instanciação de *framework* 29
Figura 2 – *AttributeFigure* e seu relacionamento com as classes que definem as figuras, como a *RectangleFigure*. 31
Figura 3 – Trecho de código dos *hot-spots AttributeFigure* e *RectangleFigure*. . . . 32
Figura 4 – *CreationTool* é uma implementação do padrão de projeto Prototype (GAMMA et al., 1995), utilizado para criar ferramentas que desenham as figuras. 33
Figura 5 – Código da classe *JavaDrawApp* específica da aplicação com a criação da nova figura Hexagono. 33
Figura 6 – Instanciação do framework JHotDraw 5.3 pra criação de uma nova figura Hexagono. 34
Figura 7 – Exemplos de tuplas em RSF 39
Figura 8 – Exemplos de relacionamentos no RSF 40
Figura 9 – Cobertura da abordagem para detecção de padrões de projeto (TSANTALIS et al., 2006). 41
Figura 10 – Metamodelo do livro de receitas proposto 44
Figura 11 – Resumo da Abordagem proposta 47
Figura 12 – Etapa 1 da abordagem proposta, processo da análise dinâmica. 48
Figura 13 – Cenários de execução para a característica “Criar um Retângulo” (*RectangleTool*) do *framework* JHotDraw. 50
Figura 14 – Processo de coleta e arquivamento dos rastros de execução. 52
Figura 15 – Pedaço do arquivo de rastro obtido para a característica ”Desenhar Círculo"do sistema JHotDrawApp. 53
Figura 16 – Etapa 2 da abordagem proposta, processo da análise estática. 54
Figura 17 – Etapa 3 da abordagem proposta, processos de filtragem para obter as ações e dados relacionados. 55
Figura 18 – Exemplo de RSF e Rastro para identificar criação de subclasses para extender *hot-spot*, retirado dos resultados para o *framework* JHotDraw. 58

Figura 19 – Exemplo de RSF e Rastro para identificar implementação de interface *hot-spot*, retirado dos resultados para o *framework* JHotDraw. 58
Figura 20 – Exemplo para ilustrar a hierarquia de super classe. 59
Figura 21 – Exemplo de RSF para identificar hierarquia de super classe, retirado dos resultados para o *framework* JHotDraw. 59
Figura 22 – Exemplo de RSF e Rastro para identificar redefinição de método *hot-spot*, retirado dos resultados para o *framework* JHotDraw. 60
Figura 23 – Exemplo de RSF e Rastro para identificar invocação de método *hot-spot*, retirado dos resultados para o *framework* JHotDraw. 61
Figura 24 – Exemplo de RSF e Rastro para identificar instanciação de classes *hot-spot*, retirado dos resultados para o *framework* JHotDraw. 62
Figura 25 – Etapa 4, passos para estuturar as receitas em HTML e compor o Livro de Receitas. 63
Figura 26 – Boxplot - Tempo de execução - Experimento com alunos da Instituição 1. 97
Figura 27 – Boxplot - Taxa de acerto do experimento com alunos da Instituição 1. . 98
Figura 28 – Boxplot - Resultados do experimento com alunos da Instituição 1. . . . 99
Figura 29 – Boxplot - Tempo de Execução do experimento com alunos da Instituição 2. 101
Figura 30 – Boxplot - Taxa de acerto do experimento com alunos da Instituição 2. . 102
Figura 31 – Boxplot - Resultados do experimento com alunos da Instituição 2. . . . 103
Figura 32 – Boxplot - Tempo de execução do experimento com Profissionais. 105
Figura 33 – Boxplot - Taxa de acerto do experimento com alunos Profissionais. . . 105
Figura 34 – Boxplot - Resultados do experimento com alunos Profissionais. 106
Figura 35 – Exemplo de regras específicas de um *framework* de editores gráficos, que usam o *framework* HotDraw (ORTIGOSA; CAMPO, 1999) 126
Figura 36 – Exemplo da linguagem de d*esign fragments* (FAIRBANKS; GARLAN; SCHERLIS, 2006). 129
Figura 37 – Receita descrita utilizando a linguagem RDL (OLIVEIRA; ALENCAR; COWAN, 2011) 131
Figura 38 – Exemplo deu uso da ferramenta FrUiT (BRUCH; SCHäFER; MEZINI, 2006). 134
Figura 39 – Exemplo *template* da ferramenta FUDA (HEYDARNOORI et al., 2012) 137

Lista de tabelas

Tabela 1 – Amostra de informações presentes na Receita *Criar Nova Figura* para tarefa que envolve o *hot-spot StandardDrawing* para o *framework* JHotDraw 46
Tabela 2 – Regras e filtros para obter as ações de instanciação. Legenda: X, Y e W são Classes/Interfaces. 57
Tabela 3 – Estudo sobre a cobertura: Classes *hot-spots* usadas nas soluções JHotDER e sua presença nas receitas. 67
Tabela 4 – Dados sobre os sujeitos humanos participantes do estudo preliminar sobre a utilidade das informações. 69
Tabela 5 – Estudo sobre a utilidade das informações presentes nas Receitas. 69
Tabela 6 – Amostra de artigos que apresentam experimentos controlados com sujeitos. 74
Tabela 7 – Métricas dos *Frameworks* JHotDraw e JFace, objetos de experimento. . 76
Tabela 8 – Organização dos Grupos de Sujeitos, Atividades, *Framework* e Documentação 80
Tabela 9 – Perguntas de pesquisa e algumas das perguntas dos questionários relacionadas. 90
Tabela 10 – Resultados dos experimento com alunos da Instituição 1. 97
Tabela 11 – Resultados dos experimento com alunos da Instituição 2. 101
Tabela 12 – Resultados dos experimento com Profissionais. 104
Tabela 13 – Resultados percentuais do experimento com alunos da Instituição 1. . . 108
Tabela 14 – Resultados percentuais do experimento com alunos da Instituição 2. . . 108
Tabela 15 – Resultados percentuais do experimento com profissionais. 109

Sumário

1	**INTRODUÇÃO**	**19**
1.1	**Soluções Propostas**	**22**
1.2	**Hipóteses**	**24**
1.3	**Objetivos e Contribuições**	**25**
1.4	**Publicações**	**26**
1.5	**Organização da Tese**	**26**
2	**FUNDAMENTAÇÃO TEÓRICA**	**27**
2.1	***Frameworks* Orientados a Objetos**	**27**
2.1.1	Uso de *Framework*	29
2.1.2	Vantagens e Desvantagens dos *Frameworks*	33
2.2	**Engenharia Reversa**	**35**
2.2.1	Análise Dinâmica	35
2.2.2	Análise Estática	38
2.3	**Resumo do Capítulo**	**41**
3	**CONSTRUÇÃO DE LIVROS DE RECEITAS**	**43**
3.1	**Metamodelo do Livro de Receitas**	**44**
3.2	**Etapas da Abordagem**	**47**
3.2.1	Etapa 1: Extrair Informações Dinâmicas	48
3.2.2	Etapa 2: Extrair Informações Estáticas	53
3.2.3	Etapa 3: Filtro	55
3.2.4	Etapa 4: Estruturar Receitas	62
3.3	**Resumo do Capítulo**	**64**
4	**AVALIAÇÃO PRELIMINAR DA ABORDAGEM**	**65**
4.1	**Cobertura do Livro de Receitas**	**66**
4.2	**Utilidade das Informações**	**68**

4.2.1 Conclusões Preliminares 72

5 AVALIAÇÃO EXPERIMENTAL DOS LIVROS DE RECEITA 73
5.1 Delineamento Experimental 73
5.1.1 Perguntas de pesquisa 75
5.1.2 Objetos 76
5.1.3 Desenho dos Experimentos 79
5.1.4 Sujeitos 83
5.1.5 Procedimento Experimental 86
5.1.6 Variáveis 89
5.1.7 Ameaças 91
5.1.8 Metodologia de Análise dos Dados 95
5.1.9 Aspectos Éticos 96
5.2 Resultados 96
5.2.1 Experimento com Alunos - Instituição 1 96
5.2.2 Experimento com Alunos – Instituição 2 101
5.2.3 Experimento com Profissionais 104
5.2.4 Conclusão dos Resultados 107
5.3 Discussão 108
5.3.1 Experimento Alunos - Instituição 1 108
5.3.2 Experimento Alunos - Instituição 2 110
5.3.3 Experimento com Profissionais 112
5.3.4 Conclusões dos Resultados 114
5.4 Conclusão do Capítulo 118

6 TRABALHOS RELACIONADOS 119
6.1 Problemas da Instanciação de *Frameworks* 119
6.2 Abordagens Baseadas em Especificação 125
6.3 Abordagens Baseadas em Exemplos 131

7 CONCLUSÃO 139
7.1 Trabalhos Futuros 141

REFERÊNCIAS 143

ANEXOS 149

Capítulo 1

Introdução

O reuso de software é um dos importantes objetivos da Engenharia de Software que pode melhorar a qualidade e produtividade no desenvolvimento de software (HEYDARNOORI et al., 2012). Os entusiastas do desenvolvimento orientado a objetos sugerem que um dos benefícios principais de se usar uma abordagem orientada a objetos é o reuso de código. No entanto, a experiência demonstra que, frequentemente, os objetos são pequenos e são especializados para uma aplicação específica. A compreensão e adaptação do objeto se torna mais demorada que sua reimplementação. O reuso orientado a objetos parece melhor suportado por meio das abstrações de alta granularidade, como por exemplo, os *frameworks* de aplicacação (SOMMERVILLE, 2011). *Frameworks* possuem papel importante para o desenvolvimento de software. PREE (1995) afirma que um bom *framework* interage com o sistema de forma amena, proporcionando altos índices de produtividade e qualidade para o projeto.

O processo de construção de uma aplicação por meio de um *framework* é denominado de instanciação do *framework*. Os *frameworks* promovem a reutilização, fornecendo uma arquitetura parcial para a aplicação alvo. Os elementos desta arquitetura são componentes de software semi-completos e flexíveis, especialmente montados para reduzir o esforço no desenvolvimento de novas aplicações dentro de um domínio específico (GAMMA et al., 1995). Em outras palavras, por meio da captura dos pontos comuns de um domínio de aplicação em um conjunto de classes abstratas projetadas e com colaborações bem definidas, *frameworks* permitem a reutilização, tanto no nível de código como no nível de projeto (HEYDARNOORI et al., 2012).

Frameworks fornecem características [1] de um domínio específico, que são unidades genéricas de funcionalidade. Existem diferentes definições para características na comunidade de Engenharia de Software, e para o trabalho aqui apresentado adotou-se a seguinte definição: características são entidades definidas como incrementos funcionais de programas (BATORY, 2006). Por exemplo, o *framework* JHotDraw [2] oferece implementação

[1] do inglês *features*
[2] http://www.jhotdraw.org

de um conjunto de características para a instanciação de aplicações que geram desenhos gráficos, que incluem desenhar figuras geométricas diversas, colorir figuras, agrupar figuras, remover linhas e formas, alterar tamanho das figuras, mover figura, desfazer e refazer uma ação, etc.

Frameworks orientado a objetos fornecem meios de reutilizar o projeto e um conjunto de características permanentes, conhecidos como *frozen-spots*, responsáveis por fornecer recursos imutáveis (PREE, 1995). Por outro lado, são usadas técnicas típicas da orientação a objetos (como por exemplo, criar subclasses e sobrescrever métodos) para definir os pontos de extensão, conhecidos como *hot-spot*, onde são conectados componentes que implementam as características de aplicações específicas (MARKIEWICZ; LUCENA, 2001), (GEORGAKOPOULOS; HORNICK; SHETH, 1995). Em outras palavras, a implementação de tais características em uma nova aplicação ocorre com ações de programação, tais como, criar subclasses para as superclasses (*hot-spot*) fornecidas pelo *framework*, implementar interfaces do *framework* ou chamar os serviços do *framework* adequadamente. Porém, compreender uma única classe sem entender seu contexto de colaboração tem utilidade limitada. Uma compreensão aprofundada do projeto do *framework* é necessária para o êxito da instanciação (BRUCH; SCHäFER; MEZINI, 2006).

Para amenizar este problema, uma boa documentação seria necessária, mas estas frequentemente não existem ou não são apropriadas (HEYDARNOORI et al., 2012), (ROBILLARD, 2009). Alguns *frameworks* apresentam pouca documentação que não seja o código-fonte e um conjunto de exemplos (JOHNSON, 1997). Em (HOU; WONG; HOOVER, 2005), os autores encontraram situações onde a documentação estava incompleta ou completamente perdida do tópico do documento. Outro problema encontrado é o tamanho da documentação de *frameworks*. Procurar por um pequeno pedaço de texto em um grande volume de documentação pode ser uma tarefa demorada (HOU; WONG; HOOVER, 2005). Logo, uma documentação completa, atualizada e direcionada pelo interesse dos desenvolvedores é importante para o processo de compreensão dos *frameworks*. Entretanto, a criação de documentação de alta qualidade é um desafio (BLOCH, 2001) especialmente tendo em conta a complexidade dos *frameworks* modernos (KIRK; ROPER; WOOD, 2007).

Segundo a literatura,uma maneira de aprender a instanciar um *framework* é por meio de exemplo (JOHNSON, 1997), (MAR; WU; JIAU, 2011). Um método viável para garantir a eficácia da documentação da frameworks é aumentar a documentação com exemplos de código confiáveis (ROBILLARD, 2009), (NYKAZA et al., 2002). Isto mostra a razão pela qual os *frameworks* deveriam abranger um conjunto rico de exemplos. Mesmo quando o *framework* apresenta documentação para instanciação, como tutoriais, os desenvolvedores muitas vezes preferem olhar para exemplos de implementação reais para uma orientação concreta. Exemplos claros e funcionais são um complemento importante para descrições textuais. Exemplos tornam o *framework* mais concreto, torna mais fácil

a compreensão do fluxo de controle, e ajuda o usuário a determinar se entendeu o resto da documentação.

Um dos fatores que motiva o uso de exemplos é que o código fonte é uma informação não ambígua e atualizada do sistema. Além disso, dependendo do *framework*, pode-se contar com muitas aplicações de código aberto já disponíveis na Internet em repositórios como OpenHub [3], Sourceforge [4] e GitHub [5], e ferramentas para pesquisa em repositórios como Code Search da Google [6] e BOA [7] (DYER et al., 2015). Além destas fontes, existem fóruns que auxiliam no aprendizado e compartilhamento de soluções, como o Stack Overflow [8]. Rocha e Maia 2016 indicam que estes fóruns podem fornecer informações que permitem redocumentar API (*Application Programming Interface*) de forma automática.

Porém, a análise manual de um conjunto, mesmo que pequeno, de exemplos de código é um processo trabalhoso e propenso a erros (COTTRELL et al., 2009), dado o tamanho significativo dos *frameworks* mais relevante, a complexidade de suas interfaces, e ao grande número de instanciações possíveis. Além do mais, analisar a implementação de características em exemplos de instanciação pode ser difícil porque as mesmas podem estar espalhadas ou entrelaçadas em um código (ROBILLARD; WEIGAND-WARR, 2005).

Seguindo esta direção, os usuários muitas vezes precisam ler grandes porções de código que não estão relacionados com a compreensão do uso do *framework*, por exemplo, código de estruturas internas do *framework* ou código específico para uma única instanciação do *framework* (BRUCH; SCHäFER; MEZINI, 2006). Além do mais, os exemplos podem apontar as características ao nível de usuário que o *framework* fornece, mas eles não vão explicar como esses recursos são fornecidos ou como o projeto da aplicação os usa (JOHNSON, 1992). Portanto, outras informações se fazem necessárias para esclarecer as características que podem ser instanciadas com o *framework*. Então, mesmo quando se pensa no código existente como uma documentação rica, o processo de aprendizado pode ser custoso. o que motiva a criação de uma documentação que organize os interesses de instanciação e apresente exemplos.

Uma possível solução para amenizar os problemas acima seria a existência de uma documentação que contivesse informações suficientemente necessárias para guiar a instanciação de características do domínio de aplicação do *framework*, contendo exemplos de código fonte extraídos do próprio *framework* e de instanciações existentes. Este documento poderia guiar o desenvolvedor na instanciação de características de interesse, apresentando exemplos destas características já implementadas em outras aplicações de forma a mitigar problemas de compreensão.

[3] https://www.openhub.net/
[4] http://www.sourceforge.net
[5] https://github.com/
[6] http://www.google.com/codesearch
[7] http://web.cs.ucla.edu/ shyoo1st/boa/
[8] http://stackoverflow.com/

1.1 Soluções Propostas

Nas últimas décadas foram propostas uma ampla variedade de técnicas de documentação de software para apoiar a compreensão e uso de *frameworks*. KRASNER E POPE (1988) construíram um cookbook com receitas para *frameworks* baseados na arquitetura MVC (*Model View Controller*) que descreve o *framework* textualmente e apresentam exemplos de código. As receitas são descritas textualmente e não apresentam uma estrutura definida. Em (ORTIGOSA; CAMPO, 1999), os autores apresentam o conceito de SmartBooks, um livro de receitas com uma interface interativa. Este foi criado com base em regras instanciação fornecidas por analistas especialistas sobre o *framework*. Em (OLIVEIRA; ALENCAR; COWAN, 2011), os autores apresentam a ferramenta ReuseTool para orquestrar tarefas de instanciação de *frameworks* dentro de um processo de reutilização especificado por um especialista sobre o *framework*. Estas abordagens são bons exemplos de documentação de *frameworks*, com foco no interesse do usuário, estruturadas como "passo-a-passo"que devem ser seguidos. Estas abordagens usam o conhecimento de especialistas ou documentações existentes. Porém, nem sempre a equipe de desenvolvimento poderá contar com um especialista ou com documentação adequada sobre o *framework*.

Motivados pela falta de especialista ou documentação apropriada e a utilidade dos exemplos de instanciação, surgiram as abordagens baseadas em exemplos. Estas normalmente utilizam extração das informações necessárias para a redocumentação a partir de exemplos de instanciação (código fonte) já existentes. Como por exemplo, JIANG et al. (2007) propõem uma abordagem para redocumentação automatizada de diagramas de sequência UML da API. Neste trabalho é utilizado mineração de cenários a partir de comportamentos, traçados de um conjunto de aplicações que utilizam a API. Em outro trabalho (COTTRELL et al., 2009), os autores descrevem a ferramenta Guido que usa várias visões coordenadas para apresentar as relações entre os exemplos, a fim de auxiliar o desenvolvedor na identificação de semelhanças e diferenças em conjuntos de exemplos. Outra ferramenta criada com este proposito é a FRUIT (BRUCH; SCHäFER; MEZINI, 2006), que combina o uso de técnicas de mineração de dados com uma apresentação dependente do contexto [9], com base em instanciações existentes do *framework* que ocorrem com frequência, as regras de reutilização são extraídas e armazenadas numa base de dados. Usuários do *framework* podem então consultar automaticamente as regras que são relevantes.

Devido a necessidade de encontrar bons exemplos de uso para extrair as informações, pesquisadores têm defendido a criação de repositórios de exemplos para abrigar exemplos de uso de *frameworks* (NEAL, 1989), (YE; FISCHER; REEVES, 2000) e (HOLMES; MURPHY, 2005). Estas abordagens diferem das demais, por que os desenvolvedores podem recuperar exemplos relevantes a partir de repositórios estruturados para isso. Porém,

[9] do inglês context-dependent presentation

os desenvolvedores devem aprender uma nova linguagem de consulta e ter uma ideia de que tipo de exemplo poderá ajudá-los com a sua tarefa (MICHAIL, 2000), ou escrever o código fonte em um estilo que está em conformidade com os exemplos presentes no repositório (YE; FISCHER; REEVES, 2000). Para mitigar esta dificuldade, HOLMES e MURPHY (2005), implementaram uma abordagem e a ferramenta Strathcona que utiliza a estrutura do código em desenvolvimento para encontrar exemplos relevantes em repositórios. Nesta abordagem, o contexto estrutural que é utilizado para formar uma consulta é extraído automaticamente a partir do código escrito pelo desenvolvedor. Um desenvolvedor que deseje procurar exemplos no repositório, só precisa emitir um pedido de procura, tal como através de uma combinação de teclas, para encontrar uma lista de exemplos relacionados à estrutura de código criada por ele.

As abordagens apresentadas anteriormente apresentaram uma evolução na redocumentação de *frameworks*, sendo independentes de uma documentação prévia ou especialista e podem ser úteis para a compreensão das implementações de características do *framework*. Entretanto, estas ainda exigem que o desenvolvedor saiba pelo menos os nomes de alguns dos *hot-spots* envolvidos na implementação desejada. Além do mais, elas se tornam menos úteis se o desenvolvedor tiver apenas uma ideia em alto nível dos *hot-spots* que precisa utilizar para a instanciação da característica.

Em relação ao problema da localização dos *hot-spots* que estão ligados a uma característica desejada, poderiam ser aplicadas várias abordagens para localização de características utilizando análise dinâmica (CORNELISSEN et al., 2009), (MAIA; LAFETá, 2013), pois as mesmas não exigem um conhecimento aprofundado do *framework* ou dos exemplos para serem obtidos. Um trabalho pioneiro neste sentido (HEYDARNOORI et al., 2012) introduz uma noção de *template* para a instanciação de características de interesse. Ele apresenta a ferramenta FUDA (*Framework API Understanding through Dynamic Analysis*) que utiliza análise dinâmica para extrair os *template* dos rastros de execução de aplicações exemplo que usam o *framework*. Um *template* de implementação especifica quais pacotes do *framework* se devem importar, classes do *framework* que devem ser estendidas, quais interfaces devem ser implementadas e quais operações de chamadas devem ser realizadas. Estes *templates* podem ser usados com um sumário conciso das etapas de implementação de uma característica do *framework*.

Heydarnoori apresenta uma solução para o problema aqui tratado ao utilizar exemplos e localizar os interesses por meio de análise dinâmica das aplicações que usam o *framework*. Até onde vai o nosso conhecimento, este é o trabalho com maior semelhança ao trabalho apresentado nesta tese, porém o trabalho de Heydarnoori somente usa informações dinâmicas, enquanto nesse trabalho proporemos uma abordagem híbrida.

Conhecendo as vantagens das abordagens baseadas em exemplo, que utilizam análise estática (análise estrutural e de padrão de projeto) ou que utilizam análise dinâmica (localização de características), propomos o uso destas duas técnicas na construção se-

miautomática de **livros de receitas** para a instanciação de *frameworks* construídos a partir do código-fonte do próprio *framework* e de aplicações (exemplos de uso) existentes. A ideia central é que as instruções necessárias (quais *hot-spots* para estender, quais métodos invocar, etc.) para criar uma instância de um *framework* para características desejada podem ser obtidos por engenharia reversa (análise dinâmica e estática). Em nossa proposta, um livro de receitas é uma documentação estruturada por característica de interesse para as quais estão associadas receitas que apresentam uma lista de tarefas a serem executadas com instruções importantes sobre os *hot-spots* e exemplos de uso que visam auxiliar na compreensão do que deve ser feito para a instanciação (*Monkey See/-Monkey Do rule* (GAMMA; BECK, 2003)). Estas instruções se referem a quais classes estender, quais interfaces implementar, quais objetos instanciar, quais métodos redefinir e/ou usar, bem como exemplos destas ações, acompanhados pelos comentários de código e padrões de projeto que devem serem utilizados.

1.2 Hipóteses

Três hipóteses foram formuladas, a seguir serão apresentadas estas hipóteses com suas argumentações.

H1) *As atividades de instanciação usando os livros de receitas propostos podem apresentar tempos de execução iguais ou melhores quando comparadas ao uso de documentações tradicionais do framework.*

Argumentação: Assume-se que a maneira mais rápida de aprender como instanciar um *framework* é por meio da obtenção de experiências, exemplos de uso do mesmo. Quando o desenvolvedor utiliza livros de receita, ele obtém uma sequência de ações organizadas para a característica de interesse, além de exemplos de uso sobre os *hot-spots*. Devido ao acesso estruturado destas informações, o desenvolvedor pode obter tempos próximos ou menores quanto comparado ao uso da documentação tradicional do *framework*. Estas documentações tradicionais são desenvolvidas com frequência por analistas especialistas sobre o *framework*. Conseguir um tempo próximo ao tempo gasto usando estas documentações pode ser considerado um ganho, uma vez que o livro de receita proposto é obtido de forma semi-automática e não dependeria de especialista para os casos de *frameworks* não documentados adequadamente.

H2) *As atividades de instanciação usando os livros de receita podem apresentar taxas de acerto iguais ou melhores quando comparadas ao uso de documentações tradicionais do framework.*

Argumentação: O uso de livros de receitas propostos impacta positivamente na taxa de acerto das atividades reais de instanciação de *frameworks*, pois auxilia na localização dos interesses e fornece um "passo-a-passo" das atividades que devem ser realizadas junto à exemplos. Exemplos de código são recursos importantes para expressar a correta

aplicação de uso da API do *framework* (MARKIEWICZ; LUCENA, 2001). Portanto, as atividades de instanciação usando livros de receita poderiam apresentar taxas de acerto iguais ou melhores quando comparadas ao uso de documentações tradicionais do *framework*. Assim como argumentado para o tempo de execução, a obtenção de uma taxa de acerto próxima à taxa das documentações tradicionais pode ser considerado um ganho porque é esperado que manuais desenvolvidos por humanos ainda sejam um nível de referência a ser alcançado por abordagens automatizadas.

H3) *O uso de livros de receitas propostos devem levar o desenvolvedor a um grau de satisfação maior, quando comparado ao uso de documentações tradicionais do framework.*

Argumentação: Os livros de receita derivados a partir de código devem apresentar informações diretas sobre o código necessário a uma tarefa acompanhadas de exemplos. A apresentação de conteúdo conforme o interesse de instanciação baseado em características funcionais, poderia facilitar o processo de localização dos *hot-spots*. Se as informações forem diretas, fáceis de localizar e úteis na solução da tarefa, o grau de satisfação deveria ser maior do que quando comparada a documentação tradicional, a qual nem sempre contempla estes requisitos.

Caso estas três hipóteses sejam válidas então a abordagem para construção de livros de receitas proposta nesta tese poderia ser adotada para as situações dos problemas identificados anteriormente, onde falta documentação ou existe documentação desatualizada do *framework* que se deseja utilizar. Para os desenvolvedores de *frameworks*, esta abordagem poderia auxiliar no desenvolvimento da documentação, uma vez que os *frameworks* são testados com a implementação de sistemas exemplo, a partir dos quais os livros de receitas poderiam ser gerados.

1.3 Objetivos e Contribuições

O objetivo deste trabalho é avaliar se o modelo de livro de receitas derivado de código é uma alternativa viável como documentação para guiar o desenvolvedor durante o processo de instanciação. Em outras palavras, a documentação gerada automaticamente pode ser tão boa quanto ou até melhor do que a documentação gerada manualmente, em termos de taxa de acerto e tempo de execução das atividades que usam tais documentações.

Para avaliar a contribuição primária desta abordagem para o uso de *frameworks* propõe-se um estudo experimental com sujeitos humanos que evidencie que as receitas propostas podem guiar instanciação de *framework* tão bem quanto, ou até melhor do que documentações tradicionais geradas manualmente, impactando positivamente nas atividades reais de instanciação de *frameworks*, em termos de tempo, taxa de acerto e satisfação do usuário ao utilizar o livro de receitas.

Além disso, outras contribuições serão: *i)* a definição de uma abordagem baseada em engenharia reversa de dados estáticos e dinâmicos do *framework* a fim de obter informações

úteis referentes aos *hot-spots* do *framework* com uma taxa de cobertura que permita o livro de receitas ser útil no direcionamento da solução e levando a uma percepção de satisfação maior ao usar o livro de receitas proposto; *ii)* implementação da abordagem, criação de ferramental apropriado para a extração destas informações e aplicação dos filtros necessários.

1.4 Publicações

A seguinte publicação contém resultados relacionados a essa tese:

Raquel F. Q. Lafetá; Marcelo A. Maia; David Röthlisberger. Framework Instantiation Using Cookbook Constructed With Static and Dynamic Analysis. In 23rd IEEE International Conference of Program Comprehension (ICPC), IEEE 2015.

1.5 Organização da Tese

Esta proposta está organizada da seguinte forma:

- Capítulo 2 apresenta fundamentos sobre *frameworks* orientados a objetos e técnicas de engenharia reversa utilizadas, que auxiliam na compreensão do problema tratado e na solução proposta.
- Capítulo 3 apresenta a solução semiautomática proposta, com detalhes sobre o processo de geração de livros de receitas.
- Capítulo 4 apresenta os estudos preliminares realizados sobre cobertura das receitas e sobre a utilidade das informações presentes nestas.
- Capítulo 5 apresenta a avaliação experimental do livro de receitas proposto, com o delineamento experimental, resultados, discussões e conclusões sobre estes experimentos.
- Capítulo 6 apresenta os trabalhos relacionados, dando destaque aos principais tipos de abordagens propostas até o momento.
- Capítulo 7 apresenta a conclusão sobre o trabalho apresentado e os trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Neste capítulo apresentamos conceitos fundamentais sobre *frameworks* orientados a objetos, suas aplicações, vantagens e desvantagens. Conceitos sobre as técnicas de engenharia reversa utilizadas e as ferramentas são apresentados neste capítulo. Estas informações são importantes para compreender o problema aqui tratado e a solução apresentada nesta tese. A Seção 2.1.1 apresenta os principais conceitos sobre *frameworks*. A Seção 2.1.2 apresenta um exemplo de uso para elucidar as vantagens e dificuldades de se instanciar um *framework*. A Seção 2.1.3 apresenta as vantagens e desvantagens de se utilizar *frameworks*. A Seção 2.2 apresenta fundamentos sobre as técnicas de engenharia reversa utilizadas na abordagem aqui proposta. A Seção 2.2.1 apresenta informações sobre técnicas de análise dinâmica e a Seção 2.2.2 apresenta informações sobre análise estática.

2.1 *Frameworks* Orientados a Objetos

Um *framework* é uma aplicação semi-completa, reusável que pode ser especializada para produzir aplicações customizadas. Geralmente é representado como um conjunto de classes abstratas, que definem um padrão de interação (colaboração) dos objetos (JOHNSON, 1997). GAMMA et al. (2003) define *frameworks* com "um conjunto de classes que colaboram e que compõe um projeto reutilizável para uma classe específica de software. Um *framework* fornece orientação arquitetural dividindo o projeto em classes abstratas, e definindo as suas responsabilidades e colaborações. Um desenvolvedor utiliza um *framework* para uma aplicação particular por meio da criação de subclasses e composição das instâncias de classes do *framework*". HEYDARNOORI (2012) define que "um *framework* captura a experiência necessária para resolver problemas em um domínio específico; esconde as partes do projeto que são comuns a todas as aplicações nesse domínio; e explicita as partes que precisam ser personalizadas".

Um *framework* provê uma solução para uma família de problemas semelhantes. Usando um conjunto de classes e interfaces que mostra como decompor a família de problemas, e como objetos dessas classes colaboram para cumprir suas responsabilidades. O conjunto

de classes deve ser flexível e extensível para permitir a construção de várias aplicações, especificando as particularidades de cada aplicação. Uma das ideias-chave dos *frameworks* são as classes abstratas. Uma classe abstrata geralmente tem pelo menos uma operação não implementada, cuja implementação é adiada para suas subclasses. Ao utilizar um *framework*, o trabalho do desenvolvedor consiste em prover os elementos que são específicos para sua aplicação, ou seja, criar subclasses a partir das classes abstratas implementando as operações que foram adiadas (JOHNSON, 1997).Está é uma definição antiga, mas ainda válida de um trabalho representativo realizado por Johnson.

Uma classe abstrata também normalmente fornece parte da implementação de suas subclasses. Por exemplo, um *template method* (GAMMA et al., 1995) define o esqueleto de um algoritmo em uma classe abstrata, adiando alguns dos passos para a subclasses (GAMMA et al., 1995). Cada passo é definido como um método separado que pode ser redefinido por uma subclasse, de modo que uma subclasse pode redefinir passos individuais do algoritmo sem alterar a sua estrutura. A classe abstrata pode deixar os passos individuais não implementados (ou seja, são métodos abstratos) ou pode fornecer uma implementação padrão (PREE, 1995). Estes métodos são chamados de âncoras (*hook*). A classe concreta deve implementar todos os métodos abstratos de sua superclasse abstrata e pode implementar (ou redefinir) qualquer um dos métodos âncora. Esta também pode usar os métodos que herda de sua superclasse abstrata. Um desenvolvedor de software implementa um aplicativo, personalizando, configurando, instanciando classes fornecidas pelo *framework* de forma adequada e chamando serviços prestados por este.

Uma das propriedades dos *frameworks* é inversão de controle. Tradicionalmente, um desenvolvedor reutiliza os componentes de uma biblioteca, implementa o *loop* principal das aplicações de controle e despacha os eventos. Com o uso de *frameworks*, ocorre uma inversão de controle. Isso é chamado de inversão de controle ou o princípio de Hollywood (VLISSIDES, 1996), "don't call us, we will call you". Em geral, em um *framework*, o programa principal é reutilizado pois é implementado no *framework*. O desenvolvedor decide o que é conectado a este e pode construir novos componentes para serem conectados. O código do desenvolvedor é chamado pelo código do *framework*. O *framework* determina a estrutura geral e o fluxo de controle do programa. Portanto, um *framework* tipicamente impõe um modelo de colaboração ao qual o desenvolvedor deve se adaptar.

Cada *framework* de aplicação fornece uma API que possuem diferentes estilos de visibilidade. Os *frameworks* de caixa branca se baseiam no conceito de herança e ligação dinâmica da orientação a objetos, que permite uma subclasse reutilizar a interface e a implementação de sua superclasse. Já os *frameworks* de caixa preta estão baseados no conceito de composição de objetos onde estes não revelam detalhes internos de sua implementação, tendo-se somente acesso à interface do mesmo. Por último, os *frameworks* de caixa cinza permitem adaptação tanto por herança e ligação dinâmica, quanto por composição de componentes, sendo um hibrido das duas abordagens anteriores. Como

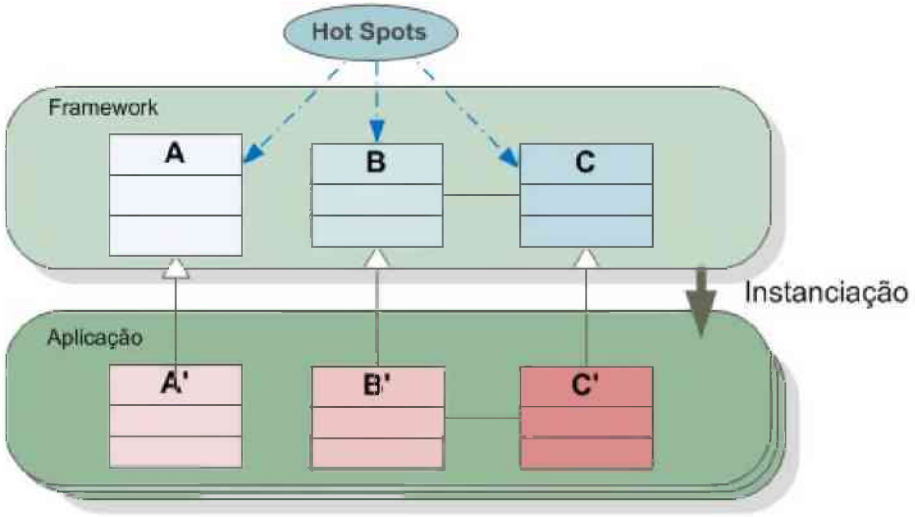


Figura 1 – Representação de uma instanciação de *framework*

os objetivos deste trabalho estão relacionados a compreensão para instanciação de *frameworks*, iremos tratar de *frameworks* de caixa branca, por estes serem mais difíceis de se compreender, uma vez que os *frameworks* de caixa preta não exigem compreensão maior sobre o *framework*.

Os *frameworks* caixa branca orientado a objetos definem um conjunto de características permanentes conhecidas como *frozen-spots* para proporcionar funcionalidades imutáveis (PREE, 1995). Ele também utiliza técnicas típicas de orientação a objetos para incorporar características flexíveis, os *hot-spots*, que devem ser estendidos e customizados de forma adequada para integrar as necessidades de aplicações específicas (MARKIEWICZ; LUCENA, 2001). Mais especificamente, *hot-spots* expressam os aspectos do domínio do *framework* que podem variar em diferentes aplicações e devem ser especializados. A API de um *framework* constitui um conjunto de *hot-spots* e estes são o principal interesse de um desenvolvedor que deseja reutilizar um *framework*. Programadores se concentram principalmente em interfaces e na interação entre o *framework* e aplicações durante a instanciação de um *framework* (HOU; WONG; HOOVER, 2005). Por este motivo, as receitas que serão propostas devem apresentar informações que auxiliem na instanciação dos *hot-spots* do *framework*, uma vez que esta é a interface para o reuso do *framework*.

2.1.1 Uso de *Framework*

Frameworks permitem que os desenvolvedores de software criem aplicativos completos com menos esforço. Alcançar esse benefício requer que o desenvolvedor use o *framework* de forma adequada: criando subclasses para as classes abstratas, instanciando objetos apropriados, e chamando os métodos de acordo com os protocolos estabelecidos (HOLMES; MURPHY, 2005). Reutilizar um *framework* é um processo humano, orientado por definição. A Figura 1 apresenta uma representação sobre como é realizada a instanciação

de *frameworks*. Cada *hot-spot* do framework deve ser ampliado com informações específicas da aplicação que é essencialmente conhecido pelos desenvolvedores de aplicativos no momento da reutilização (GEORGAKOPOULOS; HORNICK; SHETH, 1995).

Para melhorar a compreensão sobre como instanciações são realizadas, iremos apresentar um exemplo utilizando o framework JHotDraw. O JHotDraw é um framework concebido para dinamizar a implementação de aplicações para edição de desenhos. JHotDraw é uma versão Java do framework HotDraw em Smalltalk. Thomas Eggenschwiler e Erich Gamma reescreveram a aplicação HotDraw em Java como um exercício de aplicação de padrões de projeto. O JHotDraw segue uma arquitetura MVC (*Model View Control*): (a) classes baseadas no Swing (relacionadas com o controle de Visões (*view*)), (b) um controlador, e (c) as classes do modelo que são dependentes da aplicação. Uma das características do framework JHotDraw, versão 5.3, é permitir a criação de novas figuras para os editores. Por exemplo, para a criação de uma figura Hexagono, não presente na lista de figuras já implementadas no framework ou na aplicação, um usuário precisará descobrir como o framework auxilia na realização desta tarefa, o que consiste em descobrir se existem *hot-spots* no framework que ajudam na atividade de criação de uma nova figura.

Pra exemplificar, seguem alguns *hot-spots* e frozen-spots envolvidos na criação de uma figura. Vamos começar pela classe *AttributeFigure*, um *hot-spot* que usa o padrão de projeto *template method* (GAMMA et al., 1995). *AttributeFigure* mantém o controle sobre um conjunto de atributos de uma figura. Essa classe fornece um template method *draw(Graphics)* chamando *drawBackground()* seguido por *drawFrame()* (veja a Figura 2). Estes dois métodos são chamados métodos de *hook*. Eles contêm uma implementação padrão de background e a moldura para figuras, respectivamente. Estes são redefinidos em classes derivadas, tais como *RectangleFigure*, *TextFigure* e assim por diante. Normalmente, os métodos *hook* são feitos para serem redefinidos. A Figura 3, apresenta um código e demonstra esse princípio nas classes *AttributeFigure* e *RectangleFigure*. Note que a *RectangleFigure* redefine os métodos *DrawBackground()* e *drawFrame()* da *AttributeFigure*.

A forma mais comum para adicionar figuras utilizando o framework JHotDraw é utilizando a classe *CreationTool* para criar ferramentas que podem ser utilizadas para instanciar qualquer tipo de figura. *CreationTool* é um *hot-spot* que segue o padrão de projeto Prototype. *CreationTool* estende a classe *AbstractTool*, um suporte de implementação padrão para ferramentas, como se pode ver na Figura 4. *CreationTool* pode ser parametrizada para criar qualquer tipo de figura. Para criar a ferramenta, esta chama o construtor *CreationTool* e passa para ele uma instância do tipo da figura que se pretende criar.

Muitas classes e métodos do framework estão envolvidos na criação de uma figura, porém as classes do framework fornecem os métodos de serviço que podem ser chamados pelo código da aplicação e o resto dos métodos do framework são detalhes de implementação

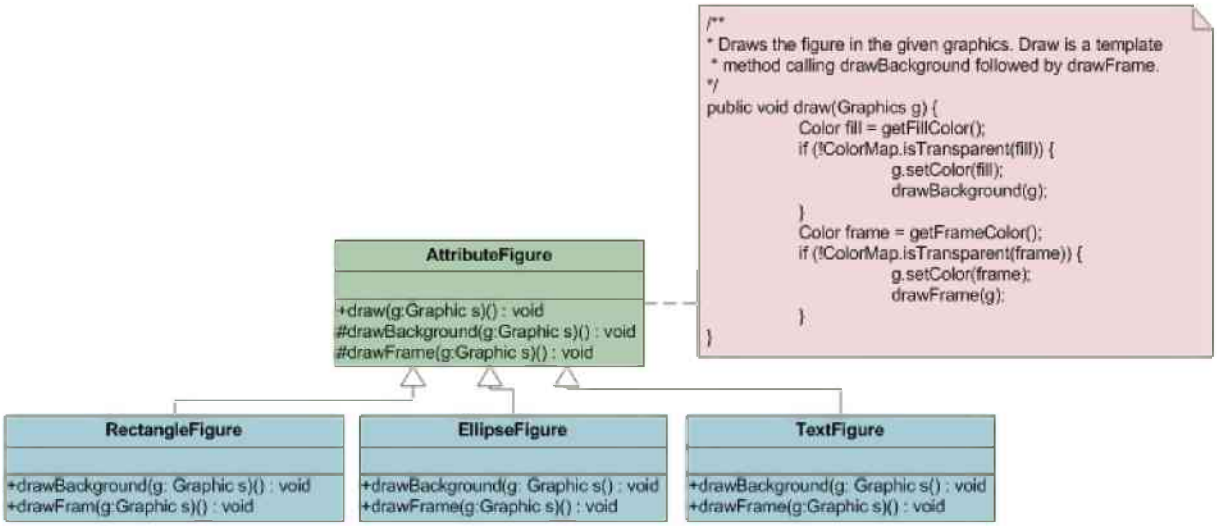


Figura 2 – *AttributeFigure* e seu relacionamento com as classes que definem as figuras, como a *RectangleFigure*.

privados (HEYDARNOORI et al., 2012). *AtributeFigure, RectangleFigure, EllipseFigure, CreationTool, CreateTool, AbtracteTool, AbstractFigure* e *Figure* são alguns exemplos de algumas das classes do framework (*hot-spots* e *frozenspots*) utilizados na criação das ferramentas que desenham figuras nas aplicações que usam o framework JHotDraw. Mas, quais *hot-spots* o desenvolvedor deverá utilizar e como utilizar? Para incorporar mais uma figura em uma ferramenta já desenvolvida que utiliza o JHotDraw (versão 5.3), como a criação da ferramenta que desenha a figura Hexagono, o desenvolvedor deverá criar uma nova classe para a figura em questão, e pode se basear nas classes *RectangleFigure*, *EllipseFigure*, *TextFigure* etc para definir a sua nova classe da aplicação. Neste exemplo, a nova classe se chamará *HexagonoFigure*, esta irá definir as formas da figura Hexagono para o desenho. Essa nova classe da aplicação poderá estender o *hot-spot RectangleFigure* ou *AttributeFigure*, ambas do framework, como uma solução. *RectangleFigure* é um *hot-spot*, implementa a figura retângulo e serve de base para a implementação de outras figuras. O desenvolvedor não precisa conhecer detalhes sobre *AttributeFigure* para implementar a atividade de criar uma nova figura que possa ser inserida pelo sistema, desde que entenda a *RectangleFigure*.

Além de criar a *HexagonoFigure*, o desenvolvedor deve na classe principal da aplicação chamar a *HexagonoFigure*. A classe principal do nosso exemplo é *JavaDrawApp* (aplicação JHotDraw 5.3). Dentro da *JavaDrawApp*, o desenvolvedor usa o método *CreationTool()*. Na classe *JavaDrawApp*, este método é redefinido para adicionar ferramentas adicionais para a criação de figuras geométricas (por exemplo: *RectangleFigure, EllipseFigure, LineConnection*), como explicamos anteriormente e é apresentado na Figura 5, contendo trechos do código da *JavaDrawApp* em que a *CreationTool* é utilizada para criar a ferramenta que desenha o Hexagono, usando a classe *HexagonoFigure*. A Figura 6 apresenta um diagrama que representa esta instanciação. Uma aplicação irá interagir

```
public abstract class AttributeFigure extends AbstractFigure {
...
/**
* Draws the figure in the given graphics. Draw is a template
* method calling drawBackground followed by drawFrame.
*/
public void draw(Graphics g) {
Color fill = getFillColor();
if (!ColorMap.isTransparent(fill)) {
g.setColor(fill);
drawBackground(g);
}
Color frame = getFrameColor();
if (!ColorMap.isTransparent(frame)) {
g.setColor(frame);
drawFrame(g);
}
}
/**
* Draws the background of the figure.
*/
protected void drawBackground(Graphics g) {
}
/**
* Draws the frame of the figure.
*/
protected void drawFrame(Graphics g) {
}
...
}

public class RectangleFigure extends AttributeFigure {
...
public void drawBackground(Graphics g) {
Rectangle r = displayBox();
g.fillRect(r.x, r.y, r.width, r.height);
}
public void drawFrame(Graphics g) {
Rectangle r = displayBox();
g.drawRect(r.x, r.y, r.width-1, r.height-1);
}
...
}
```

Figura 3 – Trecho de código dos *hot-spots* *AttributeFigure* e *RectangleFigure*.

com um framework orientado à objetos, essas interações incluem: criação de subclasses (por exemplo, *HexagonoFigure*), implementação de interfaces, redefinição de métodos da superclasse (por exemplo, *CreationTool()*), chamando métodos de serviços do framework, criando instâncias de classes do framework.

A descoberta dos *hot-spots* que se deve utilizar e como estes devem ser utilizados, normalmente é uma tarefa árdua para o desenvolvedor, que depende de uma etapa anterior que é o aprendizado do framework. Este aprendizado, como apresentado anteriormente, é um processo demorado devido a alta curva de aprendizado que envolve os *frameworks*, por serem estruturas complexas.

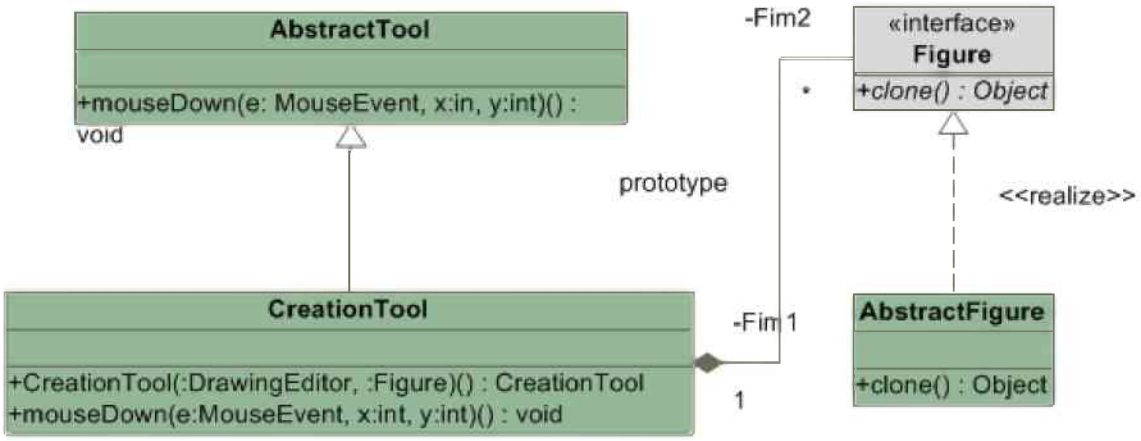


Figura 4 – *CreationTool* é uma implementação do padrão de projeto Prototype (GAMMA et al., 1995), utilizado para criar ferramentas que desenham as figuras.

```
public  class JavaDrawApp extends MDI_DrawApplication {
...
//-- DrawApplication overrides ----------------------------------------

protected void createTools(JToolBar palette) {
            super.createTools(palette);
...
            tool = new UndoableTool(new CreationTool(this, new RectangleFigure()));
            palette.add(createToolButton(IMAGES + "RECT", "Rectangle Tool", tool));
            tool = new UndoableTool(new CreationTool(this, new EllipseFigure()));
            palette.add(createToolButton(IMAGES + "ELLIPSE", "Ellipse Tool", tool));

...
            tool = new UndoableTool(new CreationTool(this, new HexagonoFigura()));
            palette.add(createToolButton(IMAGES + "HEXAGONO", "Teste tool", tool));
      }
```

Figura 5 – Código da classe *JavaDrawApp* específica da aplicação com a criação da nova figura Hexagono.

2.1.2 Vantagens e Desvantagens dos *Frameworks*

Frameworks de aplicação apresentam vantagens e desvantagens que são discutidas a seguir.

Um dos principais benefícios dos *frameworks* é que eles oferecem a reutilização de código e de projeto. Um *framework* não é simplesmente uma coleção de classes, mas também define um projeto genérico e ajuda o usuário a aplicar a arquitetura subjacente. Os problemas são resolvidos uma vez e as decisões de *design* são reutilizáveis. Em outras palavras, através da oferta de projeto reutilizáveis e código, *frameworks* **reduzem** a quantidade de decisões arquiteturais e **esforços** de implementação, teste e depuração (HEYDARNOORI et al., 2012). As aplicações desenvolvidas utilizando um *framework* usam os mesmos protocolos e compartilham a mesma arquitetura. Isto pode resultar em **melhor** capacidade de **manutenção** em cima dos *frameworks* maduros.

FAYAD et al. (1999) apresenta algumas dificuldades relacionadas ao uso de *frameworks*

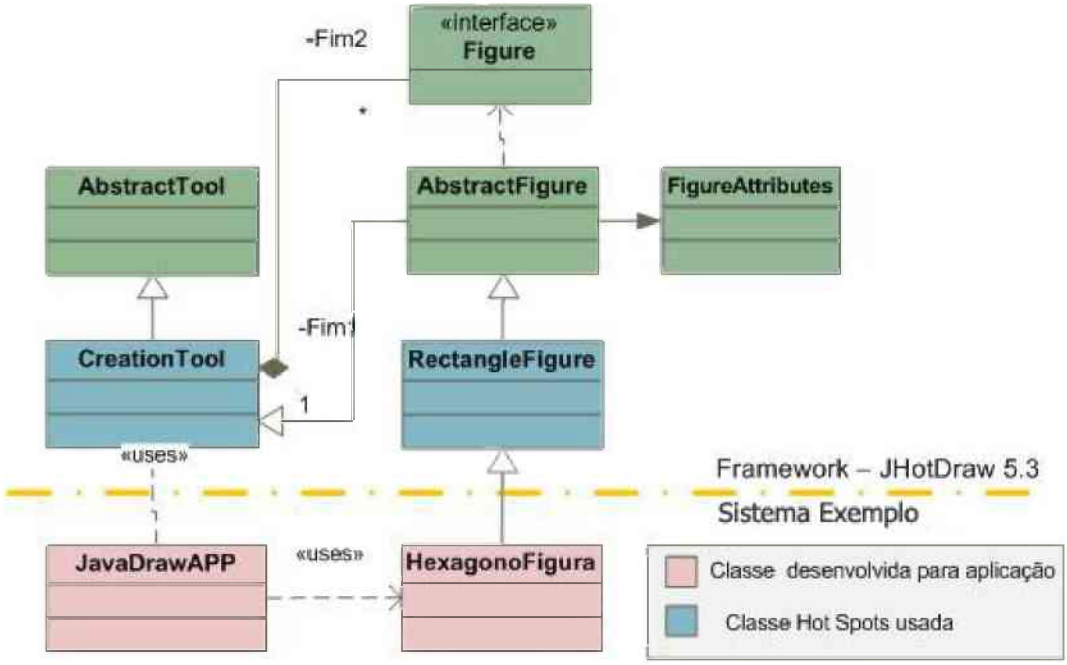


Figura 6 – Instanciação do framework JHotDraw 5.3 pra criação de uma nova figura Hexagono.

que ainda são atuais, apresentadas a seguir. As vantagens do desenvolvimento de um framework só são obtidas quando estes são reutilizados várias vezes. No entanto, é difícil prever a capacidade de reutilização de um framework de antemão. As alterações nos requisitos de domínio podem causar a desatualização do framework, e, portanto, o investimento no desenvolvimento do framework pode não ser eficaz em termos de custos. Um dos principais problemas no desenvolvimento de aplicações baseadas em framework é a curva de aprendizado íngreme (FAYAD; SCHMIDT; JOHNSON, 1999). A **complexidade, variabilidade e natureza abstrata** das APIs dos *frameworks* os tornam difíceis de aprender. Além disso, uma vez que as classes em um framework são normalmente concebidos para trabalhar em conjunto, pode ser necessário aprender várias classes simultaneamente e seus relacionamentos (HEYDARNOORI et al., 2012). *Frameworks* modernos também sofrem com a **falta de documentação adequada** que torna a situação ainda pior. Para lidar com esses problemas, autores enfatizam o papel dos exemplos de aplicações no aprendizado de *frameworks* (FAYAD; SCHMIDT; JOHNSON, 1999), (JOHNSON, 1997). Como os requisitos do framework mudam, este precisa evoluir em conformidade. Consequentemente, as aplicações desenvolvidas em cima do framework necessitam evoluir. Por isso, se o framework não é suficientemente estável, a sua evolução pode resultar em esforço de manutenção extra. Além do mais, existe uma **falta de normas** para a concepção, execução, documentação para utilizar os *frameworks* (HEYDARNOORI et al., 2012).

Esta seção apresentou informações básicas sobre *frameworks* de aplicação orientados a objeto. Um exemplo de uso de framework foi apresentado para esclarecer o processo, com suas dificuldades e vantagens. Por último, apresentou-se uma lista de vantagens

e desvantagens em se utilizar *frameworks* a fim de esclarecer o problema aqui tratado. Destacando as questões sobre a complexidade de APIs e a documentação que frequentemente é incompleta e inadequada para a compreensão do uso da API. Para lidar com este problema, o uso de exemplos de aplicações é uma abordagem frequente. Porém, esta abordagem pode ser problemática, devido ao código que implementa as características de interesse estar frequentemente espalhado ou entrelaçado no código-fonte das aplicações.

2.2 Engenharia Reversa

A engenharia reversa apoia o entendimento de um sistema através da identificação dos componentes ou artefatos do sistema, descobrindo relações entre eles e gerando abstrações dessas informações. O objetivo da engenharia reversa é não alterar o sistema de qualquer forma, mas resgatar a estrutura por traz deste. A engenharia reversa é o processo de análise de um sistema para (a) identificar os componentes do sistema e suas inter-relações e (b) criar representações do sistema em outra forma ou em um nível mais alto de abstração (CHIKOFSKY; CROSS J.H., 1990). Esta representa informações sobre o código fonte, que ajuda-nos a compreender o sistema (por exemplo, mineração de padrões de projeto) e ajuda descobrir problemas concretos no sistema (por exemplo, violação de regras e detectores de código duplicado). Ferramentas de engenharia reversa lidam basicamente com duas tarefas. A primeira tarefa é analisar código-fonte e extrair um modelo abstrato a partir do código fonte, enquanto o segundo é realizar algumas operações exploratórias neste modelo abstrato. A seguir serão apresentados conceitos sobre as duas técnicas de engenharia reversa usadas pela abordagem aqui proposta: análise dinâmica e análise estática.

2.2.1 Análise Dinâmica

A análise dinâmica é uma técnica de engenharia reversa, onde a análise dos dados é obtida a partir da execução do programa. Esta técnica tem o potencial de fornecer um quadro preciso de um sistema de software porque expõe o comportamento real e atual do sistema. As informações podem variar de detalhes sobre classes até o alto nível de visões arquiteturais. Entre os benefícios da análise estática está a disponibilidade de informações de execução e, no contexto de softwares orientados a objetos, a exposição de identidades dos objetos. A desvantagem é que a análise dinâmica só pode fornecer uma imagem parcial do sistema, ou seja, os resultados obtidos são válidos para os cenários que foram executados durante a análise. A análise dinâmica tipicamente compreende a análise da execução de um sistema através da interpretação (por exemplo, usando a máquina virtual Java) ou instrumentação, após o qual os dados resultantes são utilizados para fins como engenharia reversa e depuração. Compreensão de programas constitui uma das finalidades da analise dinâmica e, ao longo dos anos, várias abordagens de análise dinâmica têm sido

propostas neste contexto, com um amplo espectro de diferentes técnicas e ferramentas como resultado. Esta técnica tornou-se comum e tem recebido atenção considerável da comunidade de pesquisadores, em particular ao longo da última década (CORNELISSEN et al., 2009).

Ball define análise dinâmica como "a análise das propriedades de um sistema de software em execução" (BALL, 1999). Essa definição permanece propositadamente vaga, pois não especifica quais propriedades são analisadas. Isto permite que esta definição sirva em vários domínios de problemas, as propriedades exatas em análise são deixadas em aberto. Embora esta definição seja bastante abstrata, existem benefícios comuns a todas as aplicações e limitações do uso de análise dinâmica no contexto da compreensão programas, retiramos alguns do trabalho realizado por Cornelissem e colegas sobre análise dinâmica e o estado da arte (CORNELISSEN et al., 2009).

Os benefícios que consideramos são:

- Precisão no que diz respeito ao comportamento real do sistema de software, por exemplo, ocorre a resolução do mecanismo de ligação dinâmica, no contexto de software orientados a objeto, o qual pode não ser possível estaticamente.

- Técnica com estratégia orientada para o objetivo [1], o que implica na definição de um cenário de execução de tal modo que apenas as partes de interesse do sistema de software são analisados.

As limitações são:

- A inerente incompletude da análise dinâmica, como os rastros da execução em análise capturam apenas uma pequena fração do "infinito" domínio de execuções possíveis. Note-se que a mesma limitação se aplica a testes de software.

- A dificuldade de determinar quais cenários executar, a fim de disparar os elementos de interesse do programa. Na prática, conjuntos de teste podem ser usados, ou execuções envolvendo a interação do usuário com o sistema.

- A escalabilidade da análise dinâmica, devido às grandes quantidades de dados que podem ser introduzidos na análise dinâmica, e podem afetar o desempenho, armazenamento e a carga cognitiva que os seres humanos podem lidar.

- O efeito do observador, ou seja, o fenômeno em que o software age de forma diferente quando está sob observação, com um sistema intrusivo coletando os dados, pode representar um problema nos casos em que questões de tempo desempenham um papel importante.

[1] do inglês *goal-oriented strategy*

A fim de lidar com essas limitações, muitas técnicas propõem abstrações ou heurísticas, permitindo o agrupamento de pontos dos programas ou pontos de execução que compartilham certas propriedades. Em tais casos, um *trade-off* deve ser feita entre a cobertura (a cobertura é a proporção de itens que são passíveis de serem recomendados em relação ao conjunto de todos os itens conhecidos pelo sistema de recomendação) e **precisão** (a precisão de um sistema indica a quantidade de itens recomendados que são do interesse do usuário em relação ao conjunto de todos os itens que lhe são recomendados).

Um dos objetivos da análise de rastros de execução gerados pelas abordagens dinâmicas é a compreensão programas, principalmente a transmissão de informações para os seres humanos. Por este motivo, o uso de técnicas de visualização é uma abordagem popular. Uma técnica de visualização popular a este respeito é o diagrama de sequência da UML gerado usando análise dinâmica, presentes em alguns trabalhos como (PAUW et al., 2002), (SYSTä; KOSKIMIES; MüLLER, 2001), e (BRIAND; LABICHE; LEDUC, 2006). A maioria dessas abordagens oferecem certas medidas para resolver problemas de escalabilidade, como métricas e padrão de compactação. Técnicas de rastreamento de compactação populares são oferecidos por REISS e RENIERIS (2001) e HAMOU-LHADJ et al. (2004). Pensando em perspectiva de alto nível, tem havido várias abordagens para recuperação de projeto e arquitetura. Entre estes esforços, estão os artigos influentes de HEUZEROTH et al. (2002, 2003), que combinam análises estáticas e dinâmicas para detectar padrões de projeto em código legado. Uma outra porção do corpo de pesquisa tem mostrado interesse pelo estudo dos aspectos comportamentais, como por exemplo a técnica proposta por KOSKINEN et al. (2006), que usa perfis comportamentais para ilustrar regras de comportamento arquiteturalmente significativos, e um artigo de COOK e DU (2002), em que as interações de *threads* são expostas em sistemas distribuídos.

Um subcampo importante para o trabalho apresentado nesta tese é a análise de características. No presente contexto, existem análises de características fundamentais dos programas, tais como os trabalhos de MAIA e LAFETÁ (2013), CORNELISSEN et al. (2011), GREEVY et al. (2005) e KOTHARI et al. (2007), e em especial a atividade de localização característica tem se tornado cada vez mais popular desde o trabalho introdutório de WILDE e SCULLY (1995). Localização de características diz respeito ao estabelecimento de relações entre características e o código-fonte, esta técnica é um interesse de investigação popular até os dias atuais. Exemplos de outras técnicas influentes incluem WONG et al. (2000) (que utiliza *slices* de execução), EISENBARTH et al. (2003) (usando análise formal de conceitos), ANTONIOL e GUEHENEUC (2006) (por meio de análises estatísticas), e POSHYVANYK et al. (2007) (usando técnicas complementares). A análise dinâmica para localização de características é utilizada na abordagem proposta nesta tese, na recuperação dos *hot-spots* que implementam as características de interesse.

Uma técnica utilizada em análise dinâmica é a *Program Slicing* que foi originalmente definida por WEISER(1982) como um subconjunto das instruções de programas execu-

táveis que preserva o comportamento original do programa em relação a um subconjunto de variáveis de interesse V em um ponto de programa p (HEYDARNOORI et al., 2012). Diversas variações de *Program Slicing* com diversas aplicações em engenharia de software foram introduzidos na literatura.

Uma particularidade em relação à análise estática é que a análise dinâmica fornece uma imagem parcial do sistema, ou seja, os resultados obtidos são válidos para os cenários que foram executados durante a análise, em relação ao atual estado do sistema. Existem duas propriedades da análise dinâmica que devem ser cuidadosamente contempladas, sendo estas a precisão das informações coletadas e a dependência das entradas no programa. Estas propriedades são importantes por permitirem limitar o escopo da análise e relacionar as variações na entrada às mudanças internas no comportamento dos programas.

A abordagem proposta nesta tese utiliza análise dinâmica para a localização de características a fim de descobrir os elementos de código (*hot-spots*, *frozen-spots* e classes específicas da aplicação) que implementam as características de interesse. Além de obter informações sobre a hierarquia de chamadas e instanciação de objetos realizados durante a execução. Para obter as informações de cada característica, uma técnica de *slicing* é utilizada para marcar o início e fim da execução das característica, demarcando quais elementos estão relacionados a esta execução.

2.2.2 Análise Estática

A análise estática do código de um programa é a análise do código sem recorrer à sua execução ou entradas. Informações estáticas incluem artefatos de software e suas relações. Em Java, por exemplo, tais artefatos podem ser classes, interfaces, métodos e variáveis. As relações podem incluir relacionamentos de extensão entre classes ou interfaces, chamadas entre métodos, e assim por diante. O processo de engenharia reversa estática também pode incluir sintaxe e tipo de controle e verificação e análise do fluxo de dados. Como a definição da análise estática também é genérica, existem muitas técnicas e ferramentas de engenharia reversa que se enquadram como análise estática. Nesta seção serão apresentadas as técnicas e ferramentas de análise estática utilizadas na abordagem proposta.

2.2.2.1 Análise Estrutural com RSF

Todas as abordagens de engenharia reversa precisam de ferramentas para extrair as informações a serem analisadas. Na abordagem proposta nesta tese, será utilizada a ferramenta Java2RSF. Uma ferramenta que analisa o código-fonte Java e cria uma árvore de sintaxe abstrata para os arquivos de origem Java, gerando o RSF (*Rigi Standard Format*). RSF é um formato de texto simples que permite uma tupla por linha contento informações estáticas do código fonte. A Figura 7 apresenta exemplos destas tuplas

```
PACKAGE packageName
FIELD fieldname
CLASS Class1
CLASS Class2
METHOD method1
METHOD method2
HAS Class1 method1
HAS Class2 method2
CALLS Class1 method1 Class2 method2
```

Figura 7 – Exemplos de tuplas em RSF

relacionais, um arquivo RSF indicando que duas classes, *Class1* e *Class2,* e dois métodos, *method1* e *method2*, existem, que *method1* faz parte de *Class1*, que *method2* faz parte de *Class2* e que *method1* em *Class1* chama *method2* em *Class2*.

A simplicidade do RSF é uma grande vantagem e permite criar entradas e processar saídas a partir de outras ferramentas. O RSF gerado a partir do código fonte do framework e de aplicações que usam o framework é entrada para a ferramenta desenvolvida para a abordagem proposta. Como o Java é uma linguagem de programação imperativa e o formato de entrada RSF é relacional, não é possível traduzir todo o significado do código-fonte Java para RSF. No entanto, todas as informações relacionais possíveis são incluídas na saída RSF. No momento, a abordagem proposta nesta tese não está utilizando todos os dados que são extraídos usando o RSF.

Existem relações diferentes para os diferentes tipos de Java, ou seja, CLASS, INTERFACE, ENUM e ANNOTATION-TYPE. Há também uma relação do tipo CONSTRUTOR. Os blocos de construção básicos de projetos Java tem uma hierarquia clara. Os pacotes podem conter tipos, que podem conter métodos e campos. Esta hierarquia é expressa nas relações da Figura 8, parte A. A última categoria de relações óbvias são os atributos dos blocos de construção básicos do Java. Tais atributos são, por exemplo, os tipos de campos e métodos, modificadores (public, static, etc.), anotações e superclasses, como se pode ver no exemplo da Figura 8, parte B. Quatro relações adicionais são incluídos na saída RSF, para chamadas de método, leitura e escrita dos campos e uso geral de tipos, parte C conforme apresentado na Figura 8.

A escolha do RSF para análise estática foram motivados pela facilidade de se extrair os relacionamentos utilizando este ferramental, seu formato simples de entrada e saída e por apresentar os relacionamentos necessários para obter as informações definidas para compor as receitas. As informações obtidas são relevantes para determinar o relacionamento entre as diferentes classes de um projeto de Java. Todas as instâncias de chamadas de método, campos de acesso e tipos de usos podem ser extraídos.

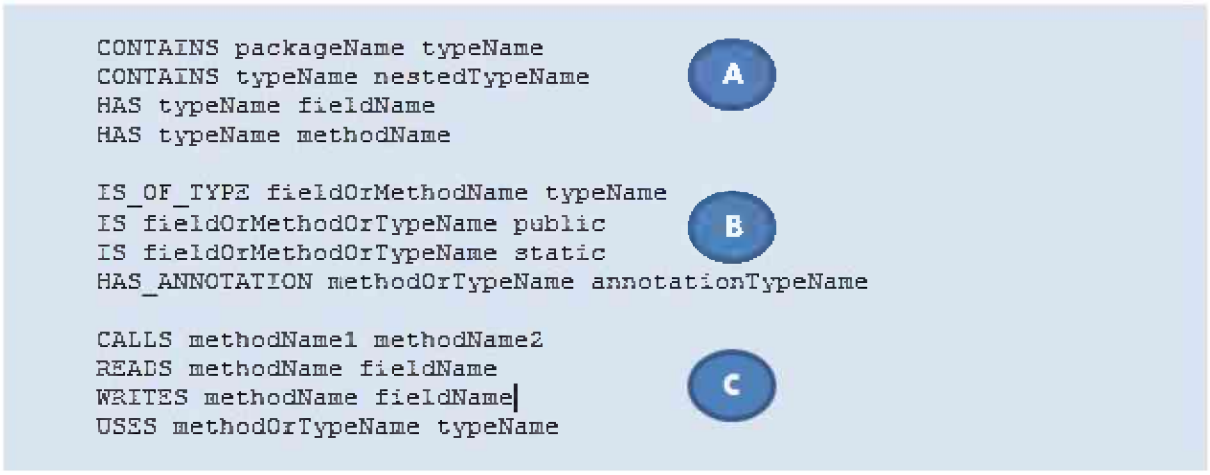


Figura 8 – Exemplos de relacionamentos no RSF

2.2.2.2 Detecção de Padrões de Projeto

Alexander diz: "Cada padrão descreve um problema que ocorre repetidamente em nosso meio, e, em seguida, descreve o núcleo da solução para esse problema, de tal forma que você pode usar esta solução um milhão de vezes, sem nunca fazer da mesma forma duas vezes" (ALEXANDER; SILVERSTEIN WITH MAX JACOBSON; ANGEL, 1977). Mesmo que Alexander estava falando sobre padrões em edifícios e cidades, o que ele diz é verdade sobre padrões de projeto de sistemas orientados a objetos. As soluções são expressas em termos de objetos e interfaces em vez de paredes e portas, mas no núcleo de ambos os tipos de padrões é a solução de um problema num contexto (JOHNSON, 1997).

Os padrões de projeto são descrições de comunicação entre objetos e classes que são customizados para resolver um problema de projeto geral em um contexto particular. O padrão de projeto identifica as classes participantes e instâncias, os seus papéis e colaborações, e a distribuição de responsabilidades. Cada padrão de projeto concentra-se em um problema de projeto particular da orientação a objetos. Ele descreve quando ele se aplica, se ele pode ser aplicado em vista de outras restrições de projeto, as consequências e as vantagens e desvantagens de seu uso. JOHNSON (1997) sugeriu padrões de projeto em linguagem natural como solução para os programadores entenderem o *frameworks*, e por este motivo, acreditando nas contribuições dos padrões de projeto para a compreensão do framework, incorporou-se tal informação a abordagem aqui proposta.

O trabalho (TSANTALIS et al., 2006) propõe um método para detecção de padrões de projeto usando similaridade de pontos (*Design Pattern Detection Using Similarity Scoring* [2]). Esta metodologia de detecção de padrão de projeto é baseada na pontuação similaridade entre vértices de gráficos. A metodologia proposta automatiza completamente o processo de detecção de padrões, extraindo as instâncias reais em um sistema para os padrões que o usuário está interessado. A aplicação da metodologia proposta em três

[2] http://java.uom.gr/ nikos/pattern-detection.html

sistemas de código aberto demonstraram a exatidão e precisão da abordagem. A Figura 9 apresenta a cobertura desta abordagem para três sistemas. Como esta abordagem foi testada com sucesso para o framework JHotDraw, nosso primeiro estudo de caso, e disponibiliza ferramental em funcionamento com documentação e uma cobertura média de 96,23%, foi a técnica selecionada para a obtenção dos padrões de projeto.

Pattern Detection Results

	JHotDraw v5.1			JRefactory v2.6.24			JUnit v3.7		
Design Patterns	*TP*	*FN*	*Recall*	*TP*	*FN*	*Recall*	*TP*	*FN*	*Recall*
Adapter/Command*	18	0	100%	7	0	100%	1	0	100%
Composite	1	0	100%	0	0	100%	1	0	100%
Decorator	3	0	100%	1	0	100%	1	0	100%
Factory Method	2	1	66.7%	1	3	25%	0	0	100%
Observer	5	0	100%	0	0	100%	4	0	100%
Prototype	1	0	100%	0	0	100%	0	0	100%
Singleton	2	0	100%	12	0	100%	0	0	100%
State/Strategy	22	1	95.6%	11	1	91.6%	3	0	100%
Template Method	5	0	100%	17	0	100%	1	0	100%
Visitor	1	0	100%	2	0	100%	0	0	100%

*Adapter refers to the Object Adapter [15].
**FP column does not exist since no false positives have been found.

Figura 9 – Cobertura da abordagem para detecção de padrões de projeto (TSANTALIS et al., 2006).

2.3 Resumo do Capítulo

Informações estáticas sobre software consistem de artefatos de software e suas relações. Em Java, por exemplo, tais artefatos podem ser classes, interfaces, métodos e variáveis. As relações podem incluir relacionamentos de extensão entre classes ou interfaces, chamadas de métodos, e assim por diante. Informações dinâmicas contêm artefatos de software também. Além disso, estas contêm informações sequenciais de rastreamento de eventos, informações sobre simultaneidade, gerenciamento de memória e vazamentos, cobertura de código, etc (SYSTA, 1999). Mesclar informações dinâmicas e estáticas em uma única visão tem vantagens e desvantagens. Se uma única documentação apresentar diretamente conexões entre informações estáticas e dinâmicas, então a qualidade da informação pode ser melhorada (SYSTA, 1999). Um exemplo disto é que se a informação estática é extraída a partir do código fonte de alguns dos artefatos, algumas relações podem ter sido ignoradas. Tal artefato pode ser, por exemplo, um construtor padrão que não está explicitamente escrito no código-fonte. Também, a chamada de tais construtores padrões pode ser ignorada. Análise dinâmica do software produz esta parte da informação, uma vez que os construtores padrão são realmente chamados quando uma instância de uma classe é criada. Em alguns casos, no entanto, a informação estática pode ser confusa quando comparada com a informação dinâmica. Por exemplo, por causa do polimorfismo, uma

chamada de método escrito no código-fonte representa um conjunto de possíveis operações, em vez de uma chamada de método que é realmente chamado em tempo de execução.

Nesta tese propomos uma abordagem que mescla análise estática e dinâmica, tendo em vista os benefícios de cada uma destas técnicas. A análise dinâmica será responsável por localizar o código que implementa as características de interesse, utilizando uma técnica de *slicing* (MAIA; LAFETá, 2013) durante a execução de cenários de execução das características. A análise estática será usada com dois propósitos, recuperar informações sobre o código fonte do framework e das aplicações que o utilizam, trazendo dados sobre as classes, interfaces e métodos, além dos relacionamentos existentes neste código. Uma técnica estática de detecção de padrões de projeto, desenvolvida por Tsantalis e colegas 2006, será utilizada para recuperar informações sobre padrões de projeto, devido a importância desta informação para a compreensão do projeto destacadas anteriormente e tanto ressaltadas por JONHSON (1997). O próximo capítulo apresentará esta abordagem proposta.

Capítulo 3

Construção de Livros de Receitas

A documentação de um *framework* deve mostrar como usá-lo para construção de aplicações. A maioria da documentação para *frameworks* descreve como este funciona em primeiro lugar, e, em seguida, descreve como usá-lo. No entanto, parece mais fácil entender conceitualmente um *framework* após alguma experiência em relação ao seu uso (JOHNSON, 1992). Na verdade, Johnson argumenta que a teoria deve seguir a prática, que uma discussão sobre a teoria subjacente a um *framework* só é compreensível uma vez que o mesmo seja entendido, e que a melhor maneira de obter tal entendimento é por meio de seu uso. A maioria dos usuários de um *framework* quer saber tão pouco quanto for possível sobre o *framework*. Isso significa que eles não estão interessados em uma descrição do projeto do *framework*, mas preferem instruções detalhadas para o uso do *framework* (JOHNSON, 1997). Esta afirmativa é evidenciada pelo crescente uso de fóruns com soluções e tutorias para uso de software como o Stack Overflow (ROCHA; MAIA, 2016) e repositórios para compartilhamento de códigos como o GitHub [1], e outros como Open Hub [2].

O livro de receitas [3] é um dos exemplos mais proeminentes de ferramentas que fornecem assistência semi-automatizada para o processo de instanciação do *framework* (ORTIGOSA; CAMPO, 1999; JOHNSON, 1997). Por definição de Krasner e Pope, livros de receitas apresentam conteúdo semi-estruturado com base em capítulos e receitas, que também tem uma estrutura típica (KRASNER; POPE, 1988). Isso não é uma idéia nova; a documentação para MacApp há muito é contido em livro de receitas e a primeira documentação para o padrão arquitetural Model View Controller (MVC) (KRASNER; POPE, 1988). Programadores iniciantes podem usar um livro de receitas para instanciar a sua primeira aplicação, e os programadores mais experientes podem usar para procurar soluções para problemas específicos.

---

[1] github.com
[2] www.openhub.net
[3] do inglês cookbook

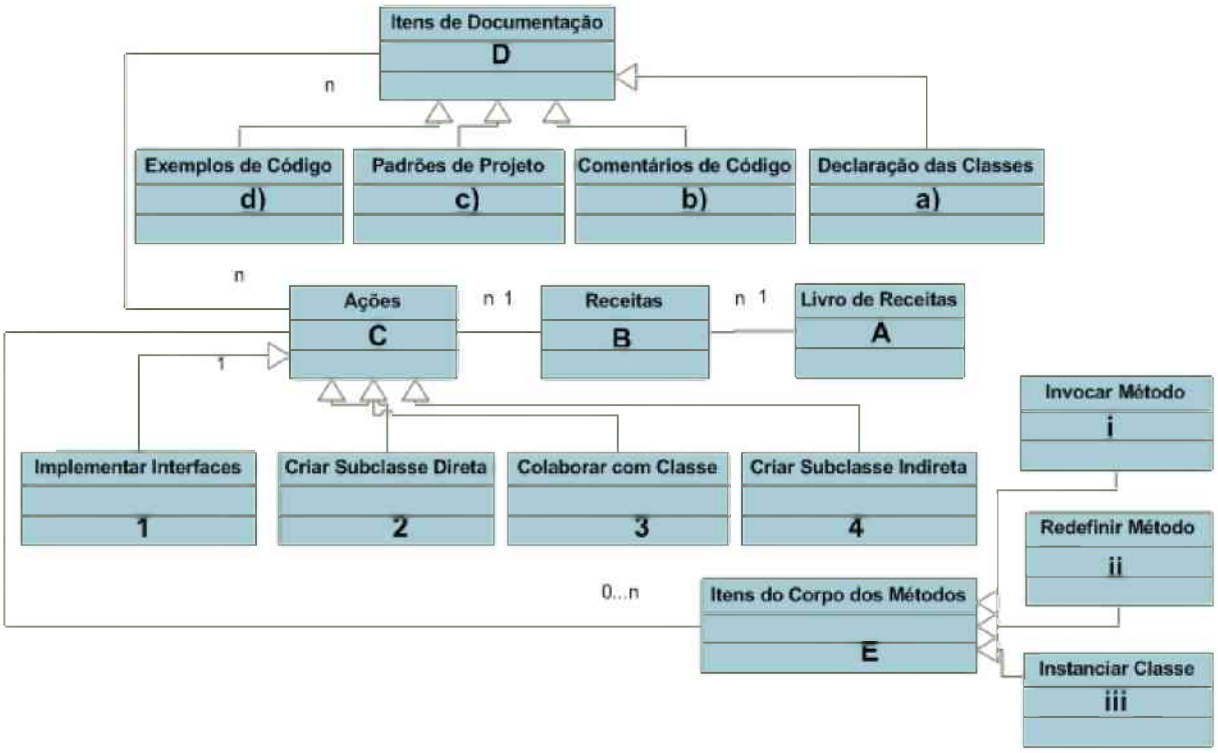


Figura 10 – Metamodelo do livro de receitas proposto

3.1 Metamodelo do Livro de Receitas

A Figura 10 apresenta o meta-modelo que serve como uma definição formal do livro de receitas proposto (item A da Figura 10) como documentação, composto por receitas (item B da Figura 10) para a instanciação de características usando o *framework*, isto é, uma lista de ações (item C da Figura 10) a serem executadas para instanciar uma característica particular do *framework*, juntamente com exemplos concretos que ilustram estas tarefas. As receitas são compostas de ações para implementar a característica. Existem quatro ações principais dentro de uma receita envolvendo os *hot-spots*, que serão apresentadas quando estas forem necessárias para a característica desejada. Para cada ação é gerada uma lista com os nomes completos (inclui pacote) dos *hot-spots* e seus comentários de código no *framework*. Segue a lista com as 4 ações e as informações:

1. criar subclasses, para estender um *hot-spot*;

2. a implementação de interface, para implementar uma interface *hot-spot*;

3. a colaboração de classes, quando ocorre a colaboração entre classes da aplicação e *hot-spots*, sem estender o *hot-spot* ou implementar interface de um *hot-spot*, (ex: instanciação de objeto).

4. a herança na hierarquia de super classe: são apresentadas classes *hot-spots* do *framework* que PODEM ser usadas por herança. Exemplo, invocação de algum método

de super classe *hot-spot*, como: A extends B e B extends C, logo A pode usar métodos de C.

Estas quatro ações principais são complementadas por uma lista com exemplos de classes específicas das aplicações que usam os *hot-spots* (do *framework*) para instanciar a característica tratada pela receita. O código fonte destes exemplos são apresentados e as suas relações de uso com os *hot-spots* (exemplo, estender o *hot-spot* X). Além de uma lista de ações secundárias ligadas aos métodos dos exemplos (item E, da Figura 10), apresentadas abaixo:

1. Redefinição de Métodos: uma lista de métodos da classe *hot-spot* que podem ser redefinidos para a característica. Para cada método, são apresentados:

   a) nome do método;

   b) parâmetros do método;

   c) comentário de código do método na classe *hot-spot*;

   d) exemplos de uso concretos (redefinição ou invocação) dos métodos extraídos das aplicações exemplo; e

   e) comentários de código dos exemplos de uso apresentados.

2. Invocação de Métodos: uma lista de métodos da classe *hot-spot* que podem ser invocados para a característica. Para cada método, são apresentados:

   a) nome do método;

   b) parâmetros do método;

   c) comentário de código do método na classe *hot-spot*;

   d) exemplos de uso concretos (redefinição ou invocação) dos métodos extraídos das aplicações exemplo; e

   e) comentários de código dos exemplos de uso apresentados.

3. instanciação de objetos desta classe *hot-spot*, para a característica desejada.

   a) classe Especifica da Aplicação Exemplo;

   b) comentário de Código da Classe Especifica;

   c) exemplo de Código para Instanciação do Objeto na Classe Especifica;

   d) comentário de Código do Exemplo.

Todas estas ações são enriquecidas com informações, que chamamos de itens de documentação (item D, da Figura 10), sobre os *hot-spot*:

1. declaração completa da classe ou interface *hot-spot*, que apresenta as dependências de um *hot-spot*.

2. comentários de código;

3. padrões de projeto a serem aplicados;

4. exemplos de uso, classes específicas das aplicações exemplo que usam o *hot-spot* para a característica desejada.

Tabela 1 apresenta parte das informações presentes na receita *Criar Figura*, para a tarefa de criar uma subclasse que estende a classe *hot-spot StandardDrawing*. O Anexo 1 apresenta partes do livro de receitas gerado para o *framework* JHotDraw e pode dar uma noção melhor sobre o conteúdo apresentado nas receitas e sua organização.

Tabela 1 – Amostra de informações presentes na Receita *Criar Nova Figura* para tarefa que envolve o *hot-spot StandardDrawing* para o *framework* JHotDraw

Tarefas e *hot-spots*:	Estender StandardDrawing
Comentário de Código - Classe ou Interface:	The standard implementation of the Drawing interface. @see Drawing
Declaração completa do *hot-spot* - Classe ou Interface:	public class StandardDrawing extends CompositeFigure implements Drawing
Padrão de Projeto - Classe ou Interface:	Design Pattern: Observer Subject: ch.ifa.draw.standard.StandardDrawing Observers: Vector fListeners. Attach(Observer):addDrawingChangeListener. Detach(Observer): removeDrawingChangeListener. Notify(): figureInvalidated, figureRequestUpdate Observer: ch.ifa.draw.framework.DrawingChangeListener Update(): drawingInvalidated, drawingRequestUpdate
Lista de *hot-spot* - Métodos:	remove(Figure figure)
Comentário de Código - Métodos:	Removes the figure from the drawing and releases it.

Exemplos de Uso - Métodos:	public synchronized Figure remove(Figure figure) {
	Figure f = super.remove(figure);
	if (f instanceof AnimationDecorator)
	return ((AnimationDecorator) f).peelDecoration();
	return f; }

Para obter estas informações, uma abordagem baseada em exemplos que utiliza análise dinâmica e estática foi desenvolvida. Esta abordagem será apresentada na próxima seção.

A análise dinâmica foi a primeira solução pensada para resolver o problema da localização dos *hot-spots* envolvidos na implementação das características de interesse, devido

a experiência obtida em trabalhos anteriores (MAIA; LAFETá, 2013). Localizar os *hot-spots* e apresentar as soluções para cada característica possibilita aos desenvolvedor ter acesso apenas as partes ligadas ao interesse do desenvolvimento, na tentativa de resolver o problema do entrelaçamento e espalhamento, direcionando assim a compreensão do desenvolvedor. Técnicas de análise estática foram incorporadas à solução, como uma forma de obter as informações estruturais dos *frameworks* e sistemas que o utilizam. Estas informações são referentes aos relacionamentos entre classes especificas da aplicação e os *hot-spots* (classes e interfaces), informações sobre os métodos do *framework* redefinidos, objetos instanciados, pacotes da aplicação envolvidos na solução, parâmetros de cada método e padrões de projeto. Além dos exemplos de código concretos.

Um novo usuário do *framework* ou um usuário experiente podem usar essa abordagem para os seus próprios fins ou para distribuir o livro de receitas gerado como um documento do *framework*. A abordagem obtém a informação de uma forma estruturada e sistemática, em vez de procurar informações no código e implementar manualmente um documento.

3.2 Etapas da Abordagem

A seguir serão apresentadas as etapas da abordagem proposta, com detalhes sobre o processo para gerar as receitas.

Figura 11 – Resumo da Abordagem proposta

O processo para a elaboração das receitas (ilustrado na Figura 11), segue 4 etapas:

1. utiliza **análise dinâmica**, por meio de rastros de execução para localizar os *hot-spots* (classes e métodos) e as classes específicas das aplicações exemplo que usam os *hot-spots* e aparecem no rastro das características executadas;

2. utiliza **análise estática** para obter as informações estruturais do *framework* e das aplicações que utilizam o *framework* e servem como exemplo. A abordagem de análise estática de TSANTALIS et al. (2006) é utilizada para obter os Padrões de Projetos dos *hot-spots*.

3. um **filtro** é realizado sobre as informações das análises estáticas e dinâmicas, apresentando apenas as informações ligadas aos *hot-spots* que implementam as características de interesse, que estão presentes nos rastros desta característica e nas relações de interesse no RSF.

4. por último, as **receitas são estruturadas** como páginas HTML com as informações obtidas nas fases anteriores, e agrupadas em um livro de receitas.

3.2.1 Etapa 1: Extrair Informações Dinâmicas

Localizar uma característica, como para a implementação de um editor de um aplicativo existente requer a identificação dos elementos de código que compõem suas extensões. Quando este editor utiliza o *framework*, cabe localizar os *hot-spots* que são utilizados. Este pode ser um desafio, porque esses pedaços de código podem não estarem localizados juntos, estarem entrelaçados com códigos de outras características, e alguns pedaços de código podem ser compartilhados entre várias características (DAGENAIS; OSSHER, 2008). Para lidar com esse problema, este trabalho propõem o uso de técnicas de localização de características utilizando análise dinâmica (MAIA; LAFETá, 2013), ou seja, rastros de execução, para localizar os conjuntos de elementos de código que implementam as características desejadas em aplicações existentes e que utilizam o *framework*.

Esses elementos de código são obtidos a partir dos rastros de execução das características, que são extraídos durante a execução de um cenário de uso que representa de forma adequada a característica. Por isso, é necessário estudar os cenários de uso adequados para obter uma cobertura boa o suficiente para ajudar na atividade de instanciação da característica. Durante o processo de extração, arquivos com rastros são gerados contento as classes executadas, eventos com chamadas de métodos e objetos instanciados. Nesta etapa são obtidas as classes criadas especificamente para a aplicação, as classes *hot-spots* (da API) e os *frozenspots* ligadas ao *framework*.

A seguir são apresentados os passos para extração destas informações dinâmicas, sumarizados na Figura 12.

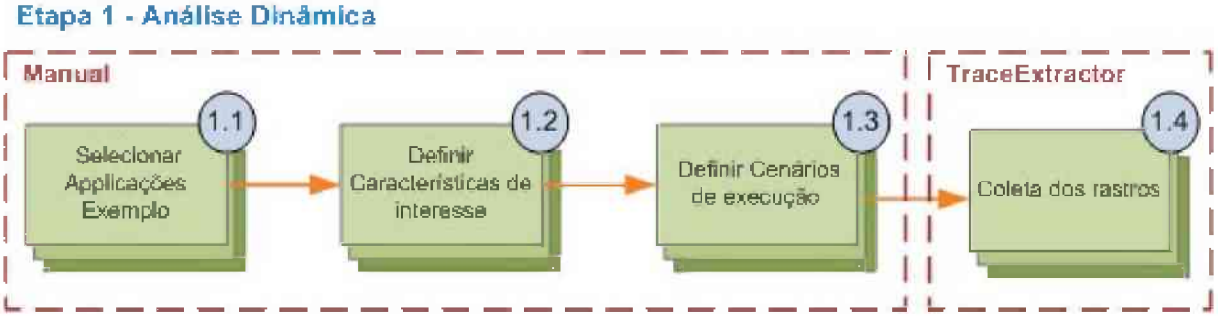


Figura 12 – Etapa 1 da abordagem proposta, processo da análise dinâmica.

Passo 1.1: Selecionar Aplicações Exemplo

Neste passo da Etapa 1, o usuário deve selecionar boas aplicações que servirão como exemplos de uso do *framework*. Modernos *frameworks* orientados a objetos normalmente vêm com uma série de exemplos de aplicações a partir do qual os desenvolvedores podem aprender sobre os *hot-spots* do *framework* (HEYDARNOORI et al., 2012).

É possível obter as informações a partir de uma única ou mais aplicações exemplos. Em (HEYDARNOORI et al., 2012), recuperou-se as informações de implementação para documentação com qualidade utilizando apenas duas aplicações de exemplo. Porém, acreditamos que quanto maior o número de aplicações usadas como exemplos, melhor os resultados obtidos. Quanto mais os contextos de aplicação das características do *framework* se diferenciam entre as aplicações, menor será a possibilidade de falsos positivos ocorrerem na análise dinâmica dos dados (HEYDARNOORI et al., 2012), conforme discutimos no próximo capítulo por meio de um estudo de caso.

Passo 1.2: Definir Características Desejada

Frameworks apresentam características definidas como uma unidade de funcionalidade que é acessível através da API do *framework* e são implementadas em aplicações que completam o código do *framework*. Ou seja, o *framework* provê uma certa funcionalidade e prescreve medidas para a sua reificação [4] e configuração em aplicações. Uma característica fornecida pelo *framework* pode ser implementada por uma ou mais classes.

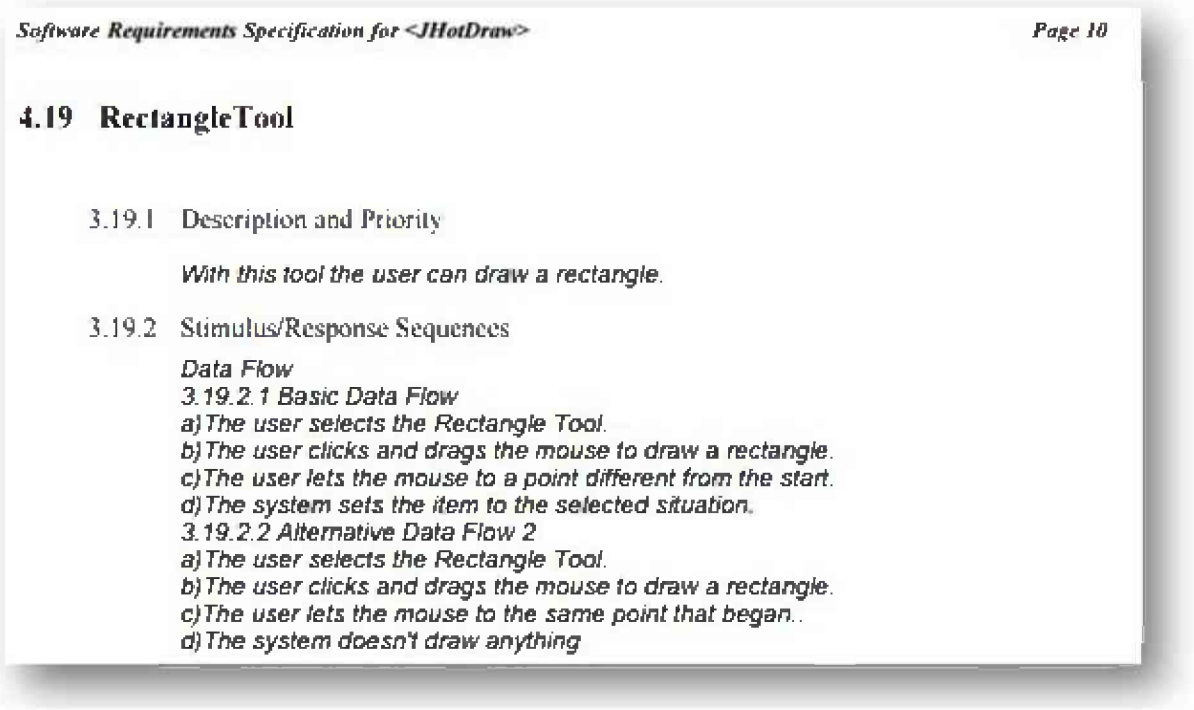
A abordagem aqui proposta, assim como outras abordagens para localização de características utilizando análise dinâmica, impõe dois requisitos em torno das características que ele pode analisar com êxito. Primeiro, o usuário da abordagem deve ter um conhecimento mínimo sobre o *framework* que fornece as características. Que consiste em conhecer as características das aplicações exemplo e como executá-las utilizando a aplicação. O usuário deve ser capaz de formular cenários de uso para as características. Em segundo lugar, deve ser possível invocar as características de interesse nas aplicações exemplo a partir de suas interfaces gráficas ou programáticas. Obedecendo estes dois requisitos, o usuário neste passo da abordagem deve definir quais características devem compor o livro de receitas, por serem de interesse para a instanciação. Este é um fator limitante para as abordagens que usam análise dinâmica.

Passo 1.3: Definir Cenários de Execução

A definição dos cenários de execução dependem da finalidade da análise e devem especificar os passos a serem seguidos para a execução do sistema em análise. No contexto específico da localização de características, os cenários devem estar ligados a esta e exercitá-las seguindo as recomendações da abordagem proposta. A análise dinâmica é sensível às entradas do programa e sua execução, portanto, deve-se fazer um planejamento prévio para as execuções, e assim poder utilizar esta análise de maneira efetiva e satisfazer os objetivos estabelecidos. Este planejamento engloba a definição adequada de cenários de uso como roteiros de execução, e a execução destes de maneira fidedigna. Os cenários

[4] reificação: atitude que consiste em tratar conceitos abstratos como se fossem reais ou objetivos

de execução podem ser expressos manualmente da mesma maneira que documentos com casos de teste são expressos. A Figura 13, apresenta um exemplo de cenário de execução, retirado de especificação de requisitos do *framework* JHotDraw [5]. Especificações e documentos com casos de teste, se existirem, podem ser usados para a elaboração dos cenários.



Software Requirements Specification for <JHotDraw> **Page 10**

4.19 RectangleTool

3.19.1 Description and Priority

With this tool the user can draw a rectangle.

3.19.2 Stimulus/Response Sequences

Data Flow
3.19.2.1 Basic Data Flow
a) The user selects the Rectangle Tool.
b) The user clicks and drags the mouse to draw a rectangle.
c) The user lets the mouse to a point different from the start.
d) The system sets the item to the selected situation.
3.19.2.2 Alternative Data Flow 2
a) The user selects the Rectangle Tool.
b) The user clicks and drags the mouse to draw a rectangle.
c) The user lets the mouse to the same point that began..
d) The system doesn't draw anything

Figura 13 – Cenários de execução para a característica "Criar um Retângulo" (*RectangleTool*) do *framework* JHotDraw.

O usuário deve garantir que o cenário execute apenas a característica ou poderá trazer informações erradas. A coleta é uma atividade humano depende, devido à execução manual, onde uma pessoa pode realizar esta atividade melhor do que a outra. O planejamento dos cenários de execução é uma atividade alheia ao uso de ferramenta nesta abordagem, dependendo unicamente do analista usuário da abordagem, e seu conhecimento sobre o sistema e o domínio da aplicação. Mas, este não é um conhecimento aprofundado sobre as aplicações ou *framework*, podendo ser facilmente adquirido. É desejável que os cenários de execução das características sejam focados na característica, deve-se tentar marcar explicitamente a invocação, onde idealmente a maioria das instruções executadas é parte da característica. Porém, ainda assim existe o risco da coleta de itens falsos positivos e falsos negativos.

Esta última regra é um desafio para abordagens que utilizam análise dinâmica. Por exemplo, para a receita que trata a criação de uma figura para o *framework* JHotDraw ("Criar Nova Figura") foi necessário coletar os rastros da seleção da figura no menu e do ato de desenhar esta figura na imagem, sendo estes dois cenários óbvios. Mas, é na

[5] http://www.randelshofer.ch/oop/jhotdraw/docs

inicialização do sistema que o menu é carregado, e portanto é este rastro que aponta para os *hot-spots* necessários para inserir esta figura no menu e qual classe trata sobre o desenho da figura selecionada. Porém a inicialização do sistema carrega diversos itens do menu que tratam de características diferentes. Para tratar este problema, a lista de receitas apresenta as características com uma descrição sobre o que foi executado para obter esta informação e como esta pode ser útil. Para este caso, criou-se uma receita para tratar a inicialização do sistema, deixando bem claro que esta receita apresenta informações sobre como os itens dos menus são carregados na aplicação. Sendo que para inserir uma nova figura na ferramenta, é necessário inserir um novo item no menu. A inicialização do sistema foi mantida separada da receita sobre "Criar Nova Figura", os usuários do livro de receitas devem ser instruídos que mais de uma receita pode ser útil para uma atividade de instanciação.

A existência de diferentes cenários de execução para a mesma característica é interessante para abordagem, quanto mais contextos diferentes forem encontrados para a mesma caraterística do *framework*, menor será a possibilidade de falsos positivos (FP) e elementos faltantes (FN - falsos negativos) nos rastros de execução. Por exemplo, para criar a receita "Criar Nova Figura" usando o *framework* JHotDraw 5.3, existem exemplos de características que criam a figura Retângulo, a figura Triângulo, a figura Círculo e a figura Polígono, dentre outras, e a coleta dos rastros destas 4 (quatro) características presentes nas aplicações que usam o *framework* irá ajudar na elaboração da receita para criar uma nova figura "qualquer", como a Hexagono do exemplo da Seção 2.1.1. Os elementos de código (classes e métodos) que são executados em comum nos rastros das 4 características é o que interessa para esta abordagem, fazendo com que falsos positivos sejam eliminados dos resultados finais. A qualidade do cenário definido e a execução de características em diferentes contextos são importantes para minimizar a ocorrência de elementos falsos negativos.

Passo 1.4: Coleta dos Rastros de Execução

Em trabalho anterior (MAIA; LAFETá, 2013), aspectos foram utilizados para a localização de características com análise dinâmica. Nesta proposta, utilizou-se esta mesma técnica baseada em aspectos para obter os elementos de código específicos das características de interesse. Dentre as razões desta escolha estão: **i)** a flexibilidade, facilidade e simplicidade da implementação; **ii)** a integração entre o aspecto responsável pela captura das informações e o sistema alvo é feita de forma transparente, podendo ser habilitada e desabilitada com facilidade; **iii)** o controle sobre as partes analisadas pode ser feitos em nível de instruções, acesso a variáveis, métodos, classes ou pacotes. A ferramenta utilizada se chama *TraceExtractor*, esta foi originalmente desenvolvida por SOBREIRA e MAIA (2008) e adaptada a para este trabalho.

A abordagem por ser baseada em exemplos para obter as informações sobre as instanciações depende de sistemas que utilizam o *framework* e apresentam bons exemplos de

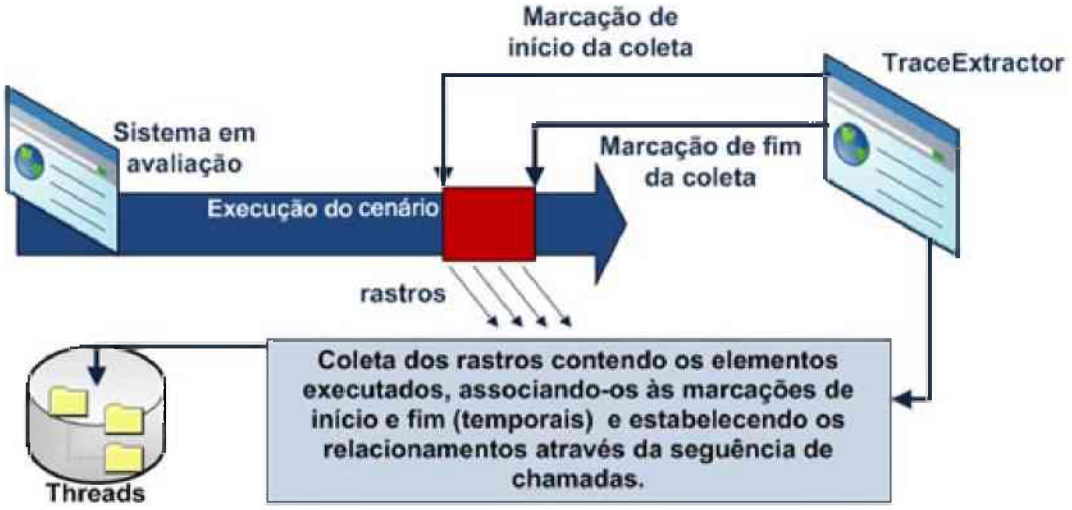


Figura 14 – Processo de coleta e arquivamento dos rastros de execução.

instanciação. Para aplicar a abordagem é necessário obter o código fonte destes sistemas. Mas, também, obter o código fonte do *framework* e incorporar este código no sistema utilizado como exemplo, caso o sistema esteja utilizando apenas o arquivos .jar. Somente com o código fonte do *framework* será possível obter na coleta de rastros informações sobre os *hot-spots* executados, utilizados para a característica desejada.

Para a coleta dos rastros, o usuário da abordagem irá compilar o sistema alvo, com o *TraceExtractor* sendo executada em paralelo e integrada a este sistema, e irá executar a(s) característica(s) desejada(s) conforme o cenário de execução predefinido, realizando a marcação do "início" e "fim" da coleta do rastro de cada característica. A *TraceExtractor*, durante a execução das características, realiza a intercepção e registro dos elementos de código que foram executados, e arquiva esta informação separada por *threads* de execução em uma pasta selecionada pelo usuário, conforme processo descrito na Figura 14. Após a coleta dos rastros de todas as características de interesse, dois tipos de arquivos serão gerados: **i)** arquivos com o rastro da execução para cada *thread* iniciada pelo sistema alvo, onde cada linha do arquivo corresponde a uma chamada de método devidamente qualificada e anotada com uma marca de tempo; e, **ii)** um arquivo de marcação dos instantes de "início" e "fim" de cada característica executada. O arquivo de rastro contempla o nome qualificado da classe, nome do método, o *timestamp* (representado por um inteiro), nível de aninhamento (chamado de *stack level* na Tabela 2) do método na pilha de chamadas e um código de identificação do objeto, método ou classe sendo chamada. A Figura apresenta um exemplo de rastro para a característica "Desenhar Círculo" do sistema JHotDrawApp.

A maneira como os rastros são coletados, notificando dos pontos de "início" e "fim" da execução das características e dando um nome para identificar esta, possibilita o corte [6], a diferenciação dos rastros das características. Por este motivo, a aplicabilidade da abor-

[6] do inglês slicing

```
CH.ifa.draw.samples.javadraw.AnimationDecorator,getDecoratedFigur
e,9,4831,1473005114570
CH.ifa.draw.figures.EllipseFigure,displayBox,9,4832,1473005114570
CH.ifa.draw.framework.FigureChangeEvent,<init>,7,4946,14730051145
70
CH.ifa.draw.samples.javadraw.BouncingDrawing,figureInvalidated,7,
1895,1473005114570
CH.ifa.draw.framework.FigureChangeEvent,getInvalidatedRectangle,8
,4946,1473005114570
CH.ifa.draw.framework.DrawingChangeEvent,<init>,8,4947,1473005114
570
CH.ifa.draw.standard.StandardDrawingView,drawingInvalidated,8,463
1,1473005114570
CH.ifa.draw.framework.DrawingChangeEvent,getInvalidatedRectangle,
9,4947,1473005114570
CH.ifa.draw.framework.FigureChangeEvent,<init>,6,4948,14730051145
70
CH.ifa.draw.samples.javadraw.BouncingDrawing,figureChanged,6,1895
,1473005114570
CH.ifa.draw.framework.FigureChangeEvent,getFigure,7,4948,14730051
14570
```

Figura 15 – Pedaço do arquivo de rastro obtido para a característica "Desenhar Círculo"do sistema JHotDrawApp.

dagem é restrita as características controláveis e observáveis pelo usuário. Uma coleta mal definida e/ou mal executada pode ocasionar problemas com elementos de código que não deveriam ser coletados e foram (falso positivos - FP) e elementos não cobertos (falsos negativos - FN) pelos cenários de execução definidos. Este tipo de problema é natural em técnicas que utilizam análise dinâmica, sendo citadas por diversos autores (WILDE; SCULLY, 1995), (LUKOIT et al., 2000), (EISENBARTH; KOSCHKE; SIMON, 2003). Mesmo com roteiros bem definidos e executados, os rastros de execução poderão conter ruídos (FP e FN) gerados pela ocorrência de eventos invisíveis a observação do usuário.

3.2.2 Etapa 2: Extrair Informações Estáticas

Na segunda etapa da abordagem, informações sobre a estrutura e código fonte das aplicações exemplo e do *framework* são obtidas utilizando análise estática. São obtidos os relacionamentos entre as classes, interfaces e métodos, quais são os elementos *hot-spots* e os específicos da aplicação, os comentários e exemplos de código. Assim como os padrões de projeto nos *hotsposts* do *framework*. A seguir serão explicados os passos desta etapa.

Passo 2.1: Obter dados Estruturais do *Framework* e Aplicações Exemplo

Neste passo da abordagem, informações estáticas sobre a estrutura do *framework* e da aplicações que usam o *framework* e servem de exemplo são obtidas. O formato padrão Rigi (RSF) e a ferramenta Java2RFS, apresentados na Seção 2.2.2.1, foram utilizados neste proposito. A ferramenta Java2RSF é executada sobre cada aplicação exemplo se-

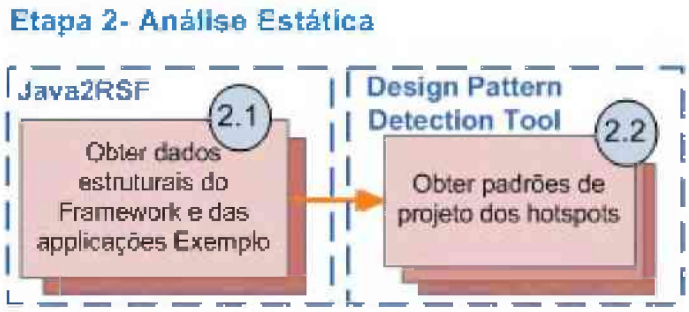


Figura 16 – Etapa 2 da abordagem proposta, processo da análise estática.

lecionada e para o *framework*. Esta extrai as informações e gera os arquivos RSF, com os elementos de código: classes, interfaces e métodos, e informações sobre os pacotes, parâmetros e tipos. Além dos relacionamentos como, tem (HAS), usa (USE), lê (READS), escreve (WRITES), estende (EXTENDS), implementa (IMPLEMENTS) e chama (CALL) apresentados para estes elementos de código.

Os arquivos RSF apresentam informações sobre todos os elementos de código das aplicações e *framework*. O interesse está nos elementos de código criados para a aplicação (elementos específicos) que usam os *hot-spots* e os próprios *hot-spots*. Conforme os pacotes referentes ao *framework* ou à aplicação identifica-se e acrescenta ao RSF informações que determinam quais são as classes do *framework* e quais são as classes da aplicação. Rotuladas com palavras chaves FRAMEWORK ou SPECIFIC, respectivamente. Com estas informações, o RSF fica pronto para ser utilizado nos filtros apresentados na próxima seção.

Passo 2.2: Obter Padrões de Projeto dos *Hot-spots*

Neste passo, os padrões de projeto utilizados nos *hot-spots* do framework são obtidos utilizando uma abordagem criado por TSANTALIS et al. (2006), este trabalho é apresentado nas Seções 6.3 e na Seção 2.2. Esta metodologia automatiza completamente o processo de detecção de padrões, extraindo as instâncias reais em um sistema para os padrões que o usuário está interessado. A aplicação da metodologia de Tsantalis e colegas sobre três sistemas de código aberto demonstrou uma acurácia e precisão da abordagem com poucos falsos negativos (elementos faltantes) e nenhum falso positivo (elementos retornados pela abordagem, mas não esperados), conforme apresentado na Figura 9.

Segundo JOHNSON (1992), a documentação de um *framework* deve descrever a finalidade do *framework*; como usá-lo; e o seu projeto detalhado. Suas afirmativas ainda são verdadeiras e ele ainda sugere que padrões de projeto como documentação são um meio para os desenvolvedores entenderem os *frameworks*, pois padrões de projeto conduzem à construção de sistemas de software bem-estruturados, de fácil manutenção, reutilizáveis e capturam a intenção por trás de um projeto, identificando objetos, suas colaborações e a distribuição de responsabilidades.

3.2.3 Etapa 3: Filtro

Na engenharia reversa a quantidade de informações extraídas normalmente é grande, contemplando milhares de linhas à serem avaliadas . Em geral, quanto mais informação anexada a um único ponto de vista, menos legível torna-se o ponto de vista, perdendo assim um dos seus principais objetivos. Nas etapas 1 e 2 são obtidos grandes volumes de informações estáticas e dinâmicas sobre as aplicações exemplos e o *framework*. Nesta etapa, um filtro é executado sobre as informações estáticas (obtidas na Etapa 2) para obter apenas dados sobre os *hot-spots* e classes específicas da(s) aplicação(ões) que utilizam os *hot-spots* para as características desejadas (obtidos no Etapa 1).

Com base nas informações obtidas pela análise dinâmica e estática, avaliou-se regras para obter as informações e ações definidas no metamodelo da Figura 10. Para trabalhar com o volume de informações obtidas na análise dinâmica e estática, desenvolveu-se a ferramenta CookFrame utilizando a linguagem JAVA. Não foi encontrada ferramenta que obtivesse as ações e informações necessárias para o metamodelo definido, por este motivo desenvolveu-se, para este trabalho, esta nova ferramenta que utiliza dados obtidos por outras já citadas, TraceExtractor e Java2RSF. Suas entradas são o RSF (resultado da Etapa 2) e os rastros (resultado da Etapa 1). Sobre as informações estáticas e dinâmicas são aplicados filtros para obter os pontos de uso da API, onde classes específicas das aplicações usam os *hot-spots*. Estes filtros obtém: **i)** as classes *hot-spot* que são usadas por alguma classe específica da aplicação; **ii)** as interfaces *hot-spots* que são usadas; **iii)** quais classes específicas das aplicações usam os *hot-spots*; **iv)** as invocações de métodos *hot-spot* (também chamados de métodos *hook* na literatura); **v)** e quais métodos são redefini por estas classes específicas; **vi)** as instanciações de objetos das classes *hot-spot* em classes da aplicação exemplo; **vii)** os pacotes que contém estes elementos de código de interesse; e **vi)** os parâmetros dos métodos. A Figura 17 ilustra este processo e a seguir serão descritas as regras para realizar estes filtro.

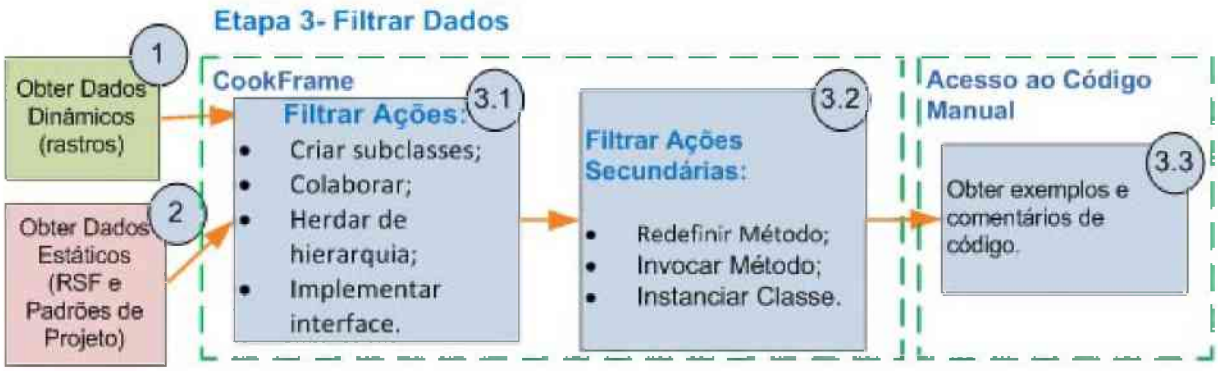


Figura 17 – Etapa 3 da abordagem proposta, processos de filtragem para obter as ações e dados relacionados.

Passo 3.1: Filtrar *Hot-spots* e Classes Específicas de Interesse

Os filtros identificam os *hot-spots* (classes, interfaces e métodos) presentes nos rastros que são usados por classes específicas da aplicação, também presentes no rastro da característica executada (Passo 3.1 da Figura 17). A identificação dos *hot-spots* e das classes da aplicação exemplo é dada pelo RSF e o filtro é realizado conforme a presença no rastro e o relacionamento desta classe específica com o *hot-spot* obtido via RSF, e se esta relação se repete no rastro. Portanto, nesta abordagem apenas considera-se os elementos de código presentes no rastro para a composição das receitas, considerados diretamente ligados à característica de interesse.

A identificação destes elementos é realizada durante a busca pelo tipo de ação do metamodelo da Figura 10: i) criar subclasse (estender); ii) implementar interfaces; iii) colaboração com classe *hot-spot*; e/ou iv) herança na hierarquia de super classe *hot-spot*. Que são complementadas pelas ações de i) instanciar classes; ii) invocar método; e/ou iii) redefinir método, que segue as regras explicadas abaixo. A Tabela 2 apresenta as regras e filtros aplicados para obter estas ações de instanciação que serão explicadas a seguir.

Para classes:

- i) Criar subclasses para estender *hot-spot*:

  Para identificar os casos de subclasses, identifica-se no RSF as relações EXTENDS entre elementos que são da aplicação (SPECIFIC no RSF) com *hot-spots* (FRAMEWORK no RSF). Se a classe da aplicação aparece no rastro para a característica, então esta ação é inserida nos resultados para gerar a receita.

  Veja o exemplo da Figura 18 e a regra simplificada do filtro na Tabela 2. Segundo o RSF a classe da aplicação AnimationDecorator (SPECIFIC) estende (EXTENDS) a classe DecoratorFigure (FRAMEWORK), abstrata do *framework*. AnimationDecorator aparece no rastro para a característica "Animar Figura", portanto essa ação de criar uma subclasse que estende a DecoratorFigure para criar a animação das figuras (movimenta as figuras aleatoriamente na tela) será inserida na receita de mesmo nome.

  Um ponto importante é que classes abstratas não aparecem no rastro, por isso não confirmamos a presença desta no rastro. O que não se faz necessário devido ao RSF e a presença da AnimationDecorator.

- ii) Implementar interface *hot-spot*:

  Para identificar os casos de implementação de interface, identifica-se no RSF as relações IMPLEMENTS entre elementos que são da aplicação (SPECIFIC) com *hot-spots* (FRAMEWORK) do *framework*. Se a classe da aplicação aparece no rastro para a característica, então esta ação é inserida nos resultados para gerar a receita.

Tabela 2 – Regras e filtros para obter as ações de instanciação. Legenda: X, Y e W são Classes/Interfaces.

Rule	Filter Condition
IMPLEMENTAR INTERFACE: X Y	Implements X Y and Specific X and Interface Y Framework Y and Trace X.anyMethod
CRIAR SUBCLASSE DIRETA: X Y	Extends X Y and Specific X and Framework Y and Trace X.anyMethod
COLABORAR COM CLASSE: X Y	Specific X and Framework Y and (Method Invocation X.methA(parA) Y.methB(parB) or Class Instantiation X:methA(parA) Y.new) and not (Extends X Y or Implements X Y or Indirect Subclassing X Y)
CRIAR SUBCLASSE INDIRETA: X Y	Specific X and Framework Y and Extends X W_1 and ... and Extends W_n Y($n \geq 1$) and Trace X.anyMethod
INVOCAÇÃO DE METODO: X.methA(parA) Y.methB(parB)	Calls X.methA(parA) Y.methB(parB) and Specific X and Framework Y and Method OF X methA(parA) AND Method OF Y methB(parB) AND Trace X.methA(parA) (n-1 stack level) AND Trace X.methB(parB) (n stack level)
REDEFINIR METODO: X.meth(par) Y.meth(par)	(Direct Subclassing X Y or Indirect Subclassing X Y) and Method OF X meth(par) and Method OF Y meth(par) and Trace X.meth(par)
INSTANCIACAO DE CLASSE: X.meth(par) Y	Calls X.methA(par) Y.new and Specific X and Framework Y and Trace X.methA(par) (n-1 stack level) and Trace Y.new (n stack level)

```
RSF:
EXTENDS CH.ifa.draw.samples.javadraw.AnimationDecorator
CH.ifa.draw.standard.DecoratorFigure
...
SPECIFIC CH.ifa.draw.samples.javadraw.AnimationDecorator
...
FRAMEWORK    CH.ifa.draw.standard.DecoratorFigure
-------------------------------------------------------------------------------
RASTRO (receita Animar Figura):
CH.ifa.draw.samples.javadraw.AnimationDecorator,<init>,6,4734,1482075599132
```

Figura 18 – Exemplo de RSF e Rastro para identificar criação de subclasses para extender *hot-spot*, retirado dos resultados para o *framework* JHotDraw.

Veja o exemplo da Figura 19, segundo o RSF a classe da aplicação BouncingDrawing (SPECIFIC) implementa (IMPLEMENTS) a classe Animatable (FRAMEWORK), interface do *framework*. BouncingDrawing aparece no rastro para a característica "Animar Figura", portanto essa ação de criar uma subclasse que implementa a interface Animatable para a característica que trata da animação das figuras no *framework* JHotDraw.

```
RSF:

IMPLEMENTS    CH.ifa.draw.samples.javadraw.BouncingDrawing
CH.ifa.draw.util.Animatable
...
SPECIFIC CH.ifa.draw.samples.javadraw.BouncingDrawing
...
FRAMEWORK    CH.ifa.draw.util.Animatable
-------------------------------------------------------------------------------
RASTRO (receita Animar Figura):

CH.ifa.draw.samples.javadraw.BouncingDrawing,<init>,5,1888,1482075590511
```

Figura 19 – Exemplo de RSF e Rastro para identificar implementação de interface *hot-spot*, retirado dos resultados para o *framework* JHotDraw.

❑ iii) Colaborar com classe *hot-spot*:

Uma classe da aplicação (SPECIFIC) pode utilizar um *hot-spot* sem ter extendido ou implementado a classe ou interface *hot-spot*. A invocação de métodos *hot-spot* e instanciação de objetos de classes *hot-spots* podem ocorrer nestas circunstancias, conforme será explicado a seguir. Quando este caso ocorrer, será qualificado na receita como uma ação de colaboração com o *hot-spot*.

❑ iv) Herdar da hierarquia de super classe *hot-spot*:

Para cobrir casos de uso do *hot-spot* devido a hierarquia de super classe, como A extends B e B extends C, logo A pode usar métodos de C. A ferramenta busca as relações de EXTENDS das superclasses de uma subclasse criada na aplicação

(SPECIFIC), veja Figura 21. Todas as classes (Hotspot1, Hotspot2, e Hotspot3 para o exemplo da Figura 20) encontradas são consideradas na busca pelas ações como métodos redefinidos, métodos invocados e instanciação de objetos, apresentadas a seguir.

```
SPECIFIC1 EXTENDS HOTSPOT1
HOTSPOT1 EXTENDS HOTSPOT2
HOTSPOT2 EXTENDS HOTSPOT3
```

Figura 20 – Exemplo para ilustrar a hierarquia de super classe.

Por exemplo, conforme o RSF, a classe JavaDrawApp é específica da aplicação exemplo e a classe MDI_ DrawApplication é do *framework*. Ainda segundo o RSF, JavaDrawApp estende MDI_DrawApplication e MDI_DrawApplication estende DrawApplication. Veja o RSF na Figura 21. Veremos a seguir que com estas informações e outras do RSF e do rastro, podemos identificar instanciações de objetos, redefinições ou invocações ligadas a esta hierarquia.

```
RSF:
EXTENDS  CH.ifa.draw.samples.javadraw.JavaDrawApp
         CH.ifa.draw.contrib.MDI_DrawApplication
---
EXTENDS  CH.ifa.draw.contrib.MDI_DrawApplication
         CH.ifa.draw.application.DrawApplication
---
SPECIFIC CH.ifa.draw.samples.javadraw.JavaDrawApp
---
FRAMEWORK CH.ifa.draw.contrib.MDI_DrawApplication
```

Figura 21 – Exemplo de RSF para identificar hierarquia de super classe, retirado dos resultados para o *framework* JHotDraw.

Passo 3.2: Obter ações secundárias

Para cada uma das ações identificadas anteriormente, se pode identificar outras ações para serem realizadas. Por exemplo, foi identificado a necessidade de criar uma subclasse estendendo a CLASSE HOTSPOT1 para a CARACTERISTICA1. Ao estender este *hot-spot*, se identifica que MÉTODO1 deve ser redefinido nesta classe específica. Seguem as regras para obter estas informações.

- i) Redefinir método:

  Para identificar os casos de redefinição de métodos de classes *hot-spot*, identifica-se no RSF as relações EXTENDS entre elementos que são da aplicação (SPECIFIC) com *hot-spots* (FRAMEWORK). Ou, olha-se a hierarquia de super classe no RSF.

Para todas as classes da aplicação (SPECIFIC) encontradas no rastro que apresente super classe, verifica-se para cada método desta classe presente no rastro, se este método possui o mesmo nome e parâmetros dos métodos das superclasses (direta ou da hierarquia) apresentados no RSF. Caso encontre um método com mesmo nome, este entra para a ação de redefinição.

No exemplo da Figura 22, segundo o RSF a classe da aplicação AnimationDecorator (SPECIFIC) estende (EXTENDS) a classe DecoratorFigure (FRAMEWORK). Pelo RSF nota-se que AnimationDecorator possui o método basicMoveBy(int,int) e que DecoratorFigure apresenta método com mesmo nome e parâmetro. No rastro, AnimationDecorator apresenta o método basicMoveBy para a característica "Animar Figura". Portanto, essa ação de redefinição de método *hot-spot* foi inserida na receita "Animar Figura".

```
RSF:
EXTENDS       CH.ifa.draw.samples.javadraw.AnimationDecorator
        CH.ifa.draw.standard.DecoratorFigure
---
METHOD CH.ifa.draw.samples.javadraw.AnimationDecorator#basicMoveBy(int,int)
---
METHOD CH.ifa.draw.standard.DecoratorFigure#basicMoveBy(int,int)
-------------------------------------------------------------------------------

RASTRO (receita Animar Figura):
CH.ifa.draw.samples.javadraw.AnimationDecorator,basicMoveBy,3,1,1473015046860
```

Figura 22 – Exemplo de RSF e Rastro para identificar redefinição de método *hot-spot*, retirado dos resultados para o *framework* JHotDraw.

❑ ii) Invocar método:

Para identificar os casos de invocação de métodos de classes *hot-spot*, identifica-se no RSF as relações CALLS entre métodos identificados no RSF como sendo de uma classe do *framework* (FRAMEWORK, METHOD_OF_HOTSPOT_NAME_METHOD) que foram chamados por uma classe da aplicação (SPECIFIC) dentro de um método X desta classe. Se a classe e o método X da aplicação (SPECIFIC) aparecem no rastro para a característica, e se o método *hot-spot* (FRAMEWORK) também aparecer no rastro ligada a esta classe da aplicação, então esta ação de invocação do método é inserida nos resultados para gerar a receita.

No exemplo da Figura 23, segundo o RSF a classe da aplicação JavaDrawApp (SPECIFIC) dentro do método createTools chama (CALLS) o método createToolButton, que segundo o RSF é um método (METHOD_OF_HOTSPOT_NAME_METHOD) do Hotspot DrawApplication (FRAMEWORK). No rastro, JavaDrawApp e o método createTools aparecem com nível de aninhamento 6 (chamado de stack level na Tabela 2), e JavaDrawApp com o método createToolButton aparece com aninha-

mento 7, logo a chamada é confirmada para a característica "Criar Nova Figura"e será inserida como uma invocação que pode ser necessária.

```
RSF:
CALLS  CH.ifa.draw.samples.javadraw.JavaDrawApp#createTools(javax.swing.JToolBar)
       CH.ifa.draw.samples.javadraw.JavaDrawApp#createToolButton(java.lang.String,ja
va.lang.String,CH.ifa.draw.framework.Tool)
---
SPECIFIC CH.ifa.draw.samples.javadraw.JavaDrawApp
---
FRAMEWORK CH.ifa.draw.application.DrawApplication
--
METHOD_OF_HOTSPOT_NAMEMETHOD CH.ifa.draw.application.DrawApplication
createToolButton

--------------------------------------------------------------------------------

RASTRO (receita Criar Nova Figura):
CH.ifa.draw.samples.javadraw.JavaDrawApp,createTools,6,1,1473005046851
CH.ifa.draw.samples.javadraw.JavaDrawApp,createToolButton,7,1,1473005046894
```

Figura 23 – Exemplo de RSF e Rastro para identificar invocação de método *hot-spot*, retirado dos resultados para o *framework* JHotDraw.

- iii) Instanciar classes *hot-spot*:

  Para identificar os casos de instanciação de classes *hot-spot*, identifica-se no RSF as relações CALLS entre elementos que são da aplicação (SPECIFIC) com *hot-spots* (FRAMEWORK), onde ocorre dentro de um método X da classe SPECIFIC a chamada de uma classe do *hot-spot* (FRAMEWORK) e esta inicializa (<init>) um objeto do FRAMEWORK. Se a classe e o método X da aplicação (SPECIFIC) aparecem no rastro para a característica, e a classe *hot-spot* (FRAMEWORK) também aparecer no rastro inicializando um objeto, então verifica-se o nível de aninhamento (*stack level* na Tabela 2) do rastro para constatar se esta classe da aplicação foi quem realmente chamou a classe do *framework*. Constatado isso, então esta ação é inserida nos resultados para gerar a receita.

  No exemplo da Figura 24, segundo o RSF a classe da aplicação JavaDrawApp (SPECIFIC) chama (CALLS) a classe CreationTool (FRAMEWORK), dentro do método createTools(javax.swing.JToolBar). Um objeto da CreationTool (<init>) é inicializado neste método. No rastro, JavaDrawApp com o seu método createTools aparecem para a característica "Criar Nova Figura". A classe CreationTool também aparece no rastro desta característica inicializando um objeto (<init>). A JavaDrawApp com seu método createTools tem nível de aninhamento 4 no rastro e a CreationTool tem nível de aninhamento 5, e estão em uma sequencia que define que JavaDrawApp chamou CreationTool pelo método createTools. Portanto essa ação de instanciação de objeto da classe CreationTool foi inserida na receita "Criar Nova Figura".

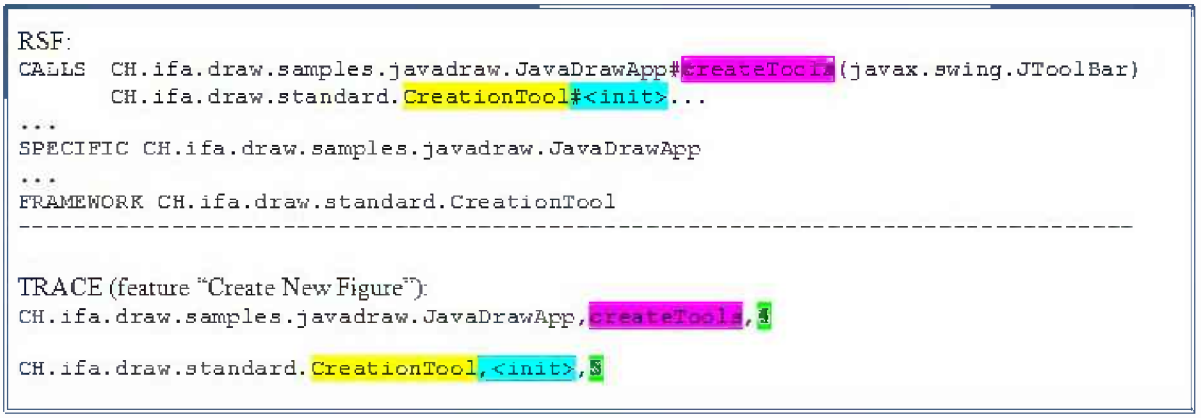


Figura 24 – Exemplo de RSF e Rastro para identificar instanciação de classes *hot-spot*, retirado dos resultados para o *framework* JHotDraw.

Ao final deste passo, a abordagem apresenta informações sobre as ações relacionadas à instanciação das características, as classes (nome completo com o pacote) e métodos *hot-spots* envolvidos nas ações e exemplos de uso (classes e métodos específicos das aplicações que usam o *framework*). Nos próximos passos, informações estáticas irão completar os dados obtidos para compor o livro de receitas.

Passo 3.3: Obter código e comentários de código

Obtidas as informações sobre as ações e elementos de código relacionado a cada ação, iremos completar com informações relativas ao código fonte e comentários de código do exemplo de uso dos *hot-spots* e dos *hot-spots* para cada ação. Estes são extraídos diretamente do código fonte manualmente, uma vez que se tem a referência sobre quais pacotes, classes e métodos destas informações. Estas informações poderiam ser obtidas de maneira automática, uma vez que se tem informações sobre o *hot-spot*, nome e pacote. Mas, esta melhoria na ferramenta ficou para trabalhos futuros.

Uma quantidade significativa de código-fonte em sistemas de software consiste de comentários, partes do código que são ignorados pelo compilador. Comentários em código representam uma fonte para a documentação de sistemas e são, portanto, úteis para a compreensão de código fonte em relação ao desenvolvimento e manutenção (STEIDL; HUMMEL; JUERGENS, 2013).

Estas informações junto com as demais são inseridas em tabelas que organizam as informações para serem convertidas automaticamente em páginas HTML estruturadas, que irão compor o livro de receitas na próxima etapa.

3.2.4 Etapa 4: Estruturar Receitas

Nesta etapa, veja Figura 25, todas as informações são organizadas em tabelas e serão utilizadas pela ferramenta CookFrame para gerar os arquivos HTML do livro de receitas. Para criar um sistema usando um *framework*, o desenvolvedor deve saber o que é

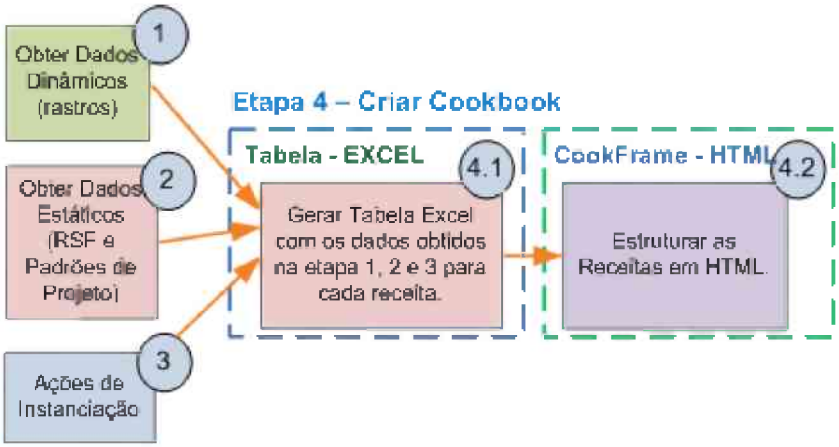


Figura 25 – Etapa 4, passos para estuturar as receitas em HTML e compor o Livro de Receitas.

possível construir com ele, portanto o livro de receitas proposto apresenta um **índice com receitas** para implementar características fornecidas pelo *framework*, este índice e fornece uma descrição pra cada receita. Receitas são estruturadas por característica, pois o desenvolvedor geralmente começa o processo de desenvolvimento de aplicações por uma lista de características desejadas. Deste modo, o desenvolvedor não precisa ler toda a documentação para a atividade alvo.

Quando o desenvolvedor escolhe uma característica para implementar, ele precisa saber quais *hot-spots* usar e como, ou seja, a compreensão das interações dentro do *framework*. Para resolver este problema, as receitas apresentam um **índice com os *hot-spots*** que podem ser usados para implementar uma determinada característica acompanhada do comentário de código de cada *hot-spot* (alguns comentário de código podem não serem úteis). A lista de *hot-spots* é dividida por **ações** (exemplo: criar subclasse, estender *hot-spot*) que podem ser necessárias para instanciar a característica utilizando estes *hot-spots*. Apenas *hot-spots* filtrados para a característica são apresentado. Ao final da lista de *hot-spots*, é apresentada uma lista de classes do sistema exemplo que são exemplos de uso destes *hot-spot* para instanciar a característica desta receita.

Ao clicar no nome de uma classe *hot-spot*, esta faz um link para outra página que apresenta mais ações (exemplo: redefinir método do *hot-spot*) que podem ser necessárias, e apresenta informações que auxiliam na compreensão do *hot-spot*. Estas ações vem acompanhadas de **informações** para orientar os desenvolvedores durante o processo de instanciação, como: **i)** comentários de código para classes e interfaces *hot-spot*; **ii)** padrões de projeto aplicados; **iii)** métodos *hot-spots* para sobrescrever; **iv)** métodos invocados; **v)** instanciação de classes, objetos criados; **vi)** comentários de código; **vii)** e exemplos de uso reais para esses métodos com base no código das aplicações existentes. Conforme se pode ver no trecho do livro de receitas apresentado no Anexo 1.

3.3 Resumo do Capítulo

Neste capítulo foi apresentada uma proposta de abordagem híbrida, que utiliza análise dinâmica e estática, para gerar as receitas que compõem o livro de receitas de forma semi-automática. O processo para geração da receita com suas etapas e passos necessários foram detalhados. O metamodelo com a estrutura do livro de receitas foi apresentado.

Capítulo 4

Avaliação Preliminar da Abordagem

Os experimentos preliminares foram realizados para avaliar a utilidade das informações presentes nas receitas e a completude e precisão da abordagem em obter estas informações. Estes resultados nos levaram a realizar melhorias na forma de apresentar as receitas e mudanças nas regras para obter uma melhor precisão e completude. As regras apresentadas no Capítulo 3 já contemplam estas melhorias na abordagem. O primeiro estudo foi elaborado para avaliar a taxa de cobertura das receitas em relação aos *hot-spots* apresentados para a primeira versão da abordagem. O segundo estudo foi elaborado para avaliar se a utilidade destas informações apresentadas nas receitas é percebida em uma atividade instanciação real por sujeitos humanos.

Os estudos preliminares foram realizados com uma versão da abordagem anterior à versão final apresentada no Capítulo 3. Nesta versão utilizou-se a ferramenta CrocoPat [1] para realizar os filtros. Na versão atual, usa-se a ferramenta CookFrame, desenvolvida pelos pesquisadores. Crocopat foi desenvolvido em 2003 por Dirk Beyer, criado para analisar modelos gráficos de projetos de software e encontrar padrões nesses gráficos. Esta ferramenta pode analisar e manipular gráficos, e também relações de aridade arbitrária. Internamente o Crocopat utiliza diagramas binários de decisão (BDD) para representar relações. Para entrada e saída dos dados, o Crocopat usa arquivos no formato RSF (*Rigi Standard Format*), que fornece as informações estáticas necessárias para a abordagem em todas as versões desta.

Para manipular as relações, Crocopat usa programas escritos na linguagem RML (*Relation Manipulation Language*). O RML é baseado no cálculo de predicados. Além das operações sobre as relações, RML também tem elementos de programação imperativa, WHILE, IF e FOR. Há também expressões para cálculos numéricos e expressões regulares para lidar com strings. Era por meio dos RMLs que se criou os filtros que obtinham as informações do livro de receitas para esta primeira versão. Tendo como entradas o RSF e os rastros que eram convertidos no formato RSF.

O Crocopat foi substituído pela ferramenta CookFrame, criada para realizar estes fil-

[1] http://www.cs.sfu.ca/dbeyer/CrocoPat

tros. O motivo foi o custo de converter e executar as informações para cada característica usando o Crocopat, e que as regras ficaram mais completas e passaram a depender do comportamento das chamadas no rastro, que não é possível ser obtido utilizando o Crocopat. A principal mudança nos filtros, entre a versão utilizada nos experimentos preliminares e a atual, é que os filtros eram mais simples e devolviam alguns *hot-spots* desnecessários nas receitas. Não considerava todas as regras apresentadas no capítulo sobre a abordagem. Por exemplo, uma dada caracteríscita para a primeira versão apresentou 24 classes *hot-spots* e não faltou nenhum *hot-spots*, por isso a cobertura foi de 100%. Esta mesma característica para a versão final, com os novos filtros, apresenta 11 classes *hot-spots* e não foram identificados falsos positivos.

Quando um *hot-spot* era identificada no rastro da característica e uma classe da aplicação que utiliza este *hot-spot* também estava no rastro desta característica, implicava na inclusão deste na receita. Na nova versão, este *hot-spot* tem que ter sido chamado no rastro por uma classe da aplicação, que também aparece no rastro; tem que ter um relacionamento confirmado com esta classe da aplicação no RSF do sistema exemplo; e esta relação tem que ser a mesma constatada no comportamento do rastro. Com a CookFrame, implementar todas estas verificações que dependem não só do RSF, mas também de dados dos Rastro, ficou mais fácil. Não seria possível com o Crocopat.

4.1 Cobertura do Livro de Receitas

Por que medir a taxa de cobertura e o que foi considerado uma boa cobertura para os livros de receita. As receitas são geradas a partir do código fonte de exemplos e foi com base nos *hot-spots* presestes e usados nestes exemplos que definiu-se a taxa de cobertura. Então a cobertura sempre dará 100% para este caso, pois está comparando com os exemplos que deram origem ao documento? A resposta é não, pois para obter as informações apresentadas nas receitas, foi realizada uma coleta de rastros para as características relevantes nos sistemas exemplo, e a partir destes rastros foram aplicados filtros para obter os comportamentos ligados somente à instanciação do *framework* e que foram executadas para o caso de uso definido. Medir a cobertura foi necessário para observar se os rastros e filtros trouxeram todos os elementos experados. Esta medida ajudou a aprimorar a abordagem, ao longo do desenvolvimento e chegar na versão final com uma boa cobertura para os dois *frameworks* objetos de pesquisa.

O primeiro estudo preliminar avaliou se as receitas contêm *hot-spots* necessários para a instanciação do *framework* (cobertura). Oito grupos de alunos, cada grupo incluindo dois estudantes de graduação em Ciência da Computação da Universidade Federal de Uberlândia, foram instruídos para instanciar o *framework* JHotDraw 5.3. Cada grupo desenvolveu um sistema chamado JHotDER que desenha diagramas de entidade relacionamento utilizando o JHotDraw.

Para gerar as receitas, foram utilizados dois sistemas que utilizam o *framework*: JHotDraw APP e JModeller [2] como exemplos de instanciação. O JHotDraw APP é um sistema exemplo que vem junto com o *framework* JHotDraw e o JModeller é citado na própria página do *framework* (http://www.jhotdraw.org/) como um bom exemplo de uso.

Tabela 3 – Estudo sobre a cobertura: Classes *hot-spots* usadas nas soluções JHotDER e sua presença nas receitas.

hot-spots (JHotDER)	% Presença nas Soluções	Presente nas Receitas?	Usado por classes específicas?	Presente nos rastros?
AttributeFigure	10%	**não**	**não**	sim
ConnectionFigure	10%	**não**	sim	**não**
LineConnection	100%	sim	sim	sim
LineFigure	100%	sim	sim	sim
MDLDrawApplication	100%	sim	sim	sim
PolyLineFigure	10%	sim	sim	sim
AbstractLineDecoration	70%	sim	sim	sim
CustomSelectionTool	100%	**não**	sim	**não**
GraphicalCompositeFigure	100%	sim	sim	sim

Todas as classes implementadas e *hot-spots* (classes e métodos) utilizados nas oito soluções JHotDER foram obtidas por meio de uma inspeção manual do código fonte das soluções. Tendo em mãos os *hot-spots* utilizados pelos alunos, verificou-se a presença desses *hot-spots* nas receitas com uma cobertura de 66% para as classes *hot-spot*. A Tabela 3 mostra que nove classes *hot-spots* foram usadas nas oito diferentes soluções do JHotDER. Três *hot-spots* (*AttributeFigure*, *ConnectionFigure* e *CustomSelectionTool*) não estavam presentes em nenhuma das receita. Destes três *hot-spots* faltantes, dois foram utilizados em apenas uma das oito aplicações JHotDER.

Sobre a cobertura dos métodos *hot-spots* nas receitas, os estudantes usaram 41 métodos *hot-spots*, 21 destes estão presentes em receitas, o que representa uma taxa de 52% de cobertura. As três classes *hot-spots* não incluídas em receitas são responsáveis pelos 48% de métodos ausentes, uma vez que a abordagem apenas lista os métodos *hot-spots* para classes incluídas nas receitas.

Dois motivos levaram a ausência destes *hot-spots* no livro de receitas: **(i)** a classe *hot-spots* não aparecem nos rastros coletados para as características selecionadas para compor o livro de receitas; **(ii)** a classe é um *hot-spots* e é abstrata, classes abstratas não são coletadas pela nossa ferramenta de coleta dos rastros. Este foi o caso da AttributeFigure. Já a *ConnectionFigure* e *CustomSelectionTool* são *hot-spots* e não foram encontrados nos rastro de execução das características presentes no livro de receitas, ou por que não foram utilizadas nos exemplos ou por falha na definição dos cenários de execução para coleta

[2] https://code.google.com/p/amock/source/browse/trunk/subjects/in/jmodeller/?r=507

dos rastros. Abordagens para localização de característica que usam análise dinâmica geralmente sofrem este problemas de cobertura (CORNELISSEN et al., 2009; MAIA; LAFETá, 2013) apresentando falsos negativos.

Este estudo foi importante para observar que a abordagem é capaz de apresentar elementos importantes para a instanciação usando o *framework*. Além de permitir observar melhorias necessárias, observando os motivos que levaram a ausência de elementos nas receitas. Com isso, melhorou-se os critérios para definição dos cenários e coletas dos rastros, e realizou-se mudanças na abordagem, que passou a usar as informações estáticas (RSF) para obter as relações e verifica-las nos rastros, o que permitiu obter inclusive as classes *hot-spots* abstratas do *framework* e melhorar a cobertura.

Neste estudo foi possível observar casos em que para uma mesma característica desejada, as aplicações exemplo podem apresentar diferentes soluções usando *hot-spots* diferentes, dependendo do contexto da aplicação. Onde os *hot-spots AbstractLineDecoration*, *CustonSelectionTool* e *GraphicalCompositeFigure* não são usados pela aplicação JHotDraw APP, mas são usados pela JModeller, e foram usados pelos grupos para criar a JHotDER. Conforme afirmado por (HEYDARNOORI et al., 2012), quanto maior for o número de características em diferentes contextos avaliadas nas aplicações exemplo para gerar as receitas, maior será a probabilidade de aumento da variedade de uso dos *hot-spots* para instanciar uma mesma característica, e se pode ter uma maior variedade de características instanciadas para opções de uso dos *hot-spots*.

4.2 Utilidade das Informações

Foi realizado um segundo estudo com cinco desenvolvedores de software graduados com experiência profissional e no uso de *frameworks*. Estes desenvolvedores fazem parte da rede de contatos dos pesquisadores. A Tabela 4 apresenta em sua segunda e terceira coluna dados relativos aos anos de experiência profissional e a quantidade aproximada de *frameworks* já instanciados por estes desenvolvedores. Foi solicitado a estes sujeitos a realização de uma instanciação do *framework* JHotDraw 5.3 para a criação de uma nova característica, a implementação de uma figura hexágono no menu de figuras que se pode desenhar com a aplicação exemplo JHotDraw APP. Os participantes receberam três documentos diferentes para apoiá-los nesta atividade: **i)** um documento com instruções sobre o experimento, arquivos para preparar o ambiente e a descrição da tarefa; **ii)** a receita relacionada com a elaboração de uma nova figura com o *framework* JHotDraw 5.3 em um formato textual; **iii)** e um questionário com 12 perguntas, 11 fechadas (com justificativa opcional) e 1 aberta.

O questionário contém questões sobre a utilidade de cada informação presente na receita e incluía também uma questão artificial de controle para verificar se os participantes responderam adequadamente às perguntas. Esta questão controle foi sobre a utilidade de

Tabela 4 – Dados sobre os sujeitos humanos participantes do estudo preliminar sobre a utilidade das informações.

Sujeitos	Anos de Experiência Profissional	Quantidade de *Frameworks* Instanciados	Sucesso na Atividade	Tempo para Executar Atividade (aproximado)
Sujeito 1	9	20	sim	2h 50min
Sujeito 2	4	10	sim	1h
Sujeito 3	10	30	sim	1h 30min
Sujeito 4	5	10	sim	1h
Sujeito 5	8	12	sim	1h 20min

métricas (número de classes, métodos, quantidade de linhas, etc) do *framework*. Esta informação é irrelevante para a instanciação de uma característica. Todos os participantes realizaram a atividade com êxito, responderam que a utilidade das métricas era fortemente inútil ou fracamente inútil, e concluíram a atividade real de instanciação dentro do prazo de três horas estipulado. Eles realizaram a atividade com um tempo médio de 1 hora e 30 minutos. No final do experimento todos enviaram os questionários respondidos e o código alterado, constando a nova funcionalidade para avaliação.

Tabela 5 – Estudo sobre a utilidade das informações presentes nas Receitas.

Sujeito	H-C/I	D-C/I	PP-C/I	CC-C/I	H-M	CC-M	PP-M	ExC-M	M
Sujeito 1	1	2	1	-1	2	2	1	2	-2
Sujeito 2	1	1	-1	0	-2	-2	-1	0	-2
Sujeito 3	1	1	2	1	1	1	2	1	-1
Sujeito 4	1	-1	-2	-2	-2	-2	-2	2	-2
Sujeito 5	1	2	1	2	1	2	2	2	-1

As informações presentes nesta receita e suas avaliações por sujeitos são apresentadas na Tabela 5. Os nomes das informações da receita são abreviados na tabela da seguinte forma:

- **H-C/I**: *hot-spots* índice (lista de classes ou interfaces);
- **D-C/I**: declaração completa da classe ou interfaces *hot-spots*;
- **PP-C/I**: Padrões de Projeto (classes e interfaces);
- **CC-C/I**: Os comentários de código (classes ou interfaces);
- **H-M**: métodos *hot-spots* para redefinir ou invocar;
- **CC-M**: Os comentários de código (métodos);

- **PP-M**: padrões de projeto (métodos);
- **ExC-M**: Exemplos de código (métodos);
- **M**: Métricas (questão controle).

Os sujeitos avaliaram as informações da receita, tal como apresentado na Tabela 5. Eles foram instruídos à classificar a utilidade conforme uma escala Likert de -2 a +2, com o seguinte significado para as diferentes classificações:

- **+2:** *Fortemente úteis:* Informação considerada como muito útil e completa (o participante não precisa de informações externas, como ter de olhar no código para entender o que deve ser feito);
- **+1:** *Fracamente útil:* A informação é considerada útil, mas não completa;
- **0:** *Visão Neutra:* A Informação não é útil e nem inútil;
- **-1:** *Fracamente inútil:* A informação não foi muito útil para o assunto;
- **-2:** *Fortemente inútil:* O sujeito considera a informação como completamente inútil.

A maioria dos indivíduos considerou como úteis (escala de Likert +2 e +1) as informações da receita, com exceção da informação com os comentários de código das classes ou interfaces *hot-spots* (CC-C/I), que tiveram uma classificação neutra, com um voto para cada nível de classificação Likert. Esta avaliação pode ter decorrido do fato que as informações presentes nos comentários de código das classes (já os comentários de código dos métodos foram bem qualificados) também estavam presentes no índice com o as classes ou interfaces *hot-spots* (H-C/I) que contempla o nome do *hot-spot* e o comentário de código sobre sua utilidade.

O índice com os *hot-spots* (H-C/I), a declaração completa da classe e interface *hot-spot* (D-C/I) e os exemplos de código dos métodos (ExC-M) apresentaram os resultados mais significativos com, respectivamente, **100%** (5 dos 5), **80%** (4 dos 5) e **80%** dos participantes classificando estas informações como úteis. Para este estudo, todos os *hot-spots* utilizados na atividade foram descritos na receita, o que representa uma taxa de **100%** da cobertura. Fomos mais cautelosos com a coleta dos rastros, executamos um número maior (um total de seis) de características diferentes ligadas a criação de figuras diversas para a criação desta receita.

Junto à cada questão para classificação na escala de utilidade havia um campo para justificar as respostas, sendo este opcional. Estas justificativas auxiliaram em uma avaliação qualitativa dos resultados, foram úteis para direcionar a compreensão sobre a avaliação das informações. Alguns participantes inclusive utilizaram este campo para dar sugestões de melhorias para a abordagem. Por exemplo, as informações sobre padrões de projeto

(PP-C/I e PP-M) foram consideradas fortemente úteis (+2) pelo Sujeito 3 (veja Tabela 5), e fortemente inútil pelo Sujeito 4. O Sujeito 3 é o mais experiente do grupo. O Sujeito 4 não usou a informação sobre o padrão de projeto, isto foi informado por ele neste campo de justificativa. Já o Sujeito 3 utilizou o padrão de projeto e justificou: "Ganha-se muito tempo sabendo quais classes são implementações de padrões de projeto já conhecidos, facilitando que a nova implementação não quebre a arquitetura do código atual". O Sujeito 4, foi o que pior qualificou as informações, mas ele justifica da seguinte maneira: "A informação estava completa no exemplo", ele usou o exemplo de código (ExC-M) e foi direcionado pelo índice com os *hot-spots* (H-C/I). Com estas duas informações, o Sujeito 4 implementou a característica. Uma observação interessante, mesmo que em uma população pequena, é que os analistas com maior experiência profissional (ver Tabela 4), são os que melhor qualificaram a utilidade das informações presentes na receita e que gastaram um tempo maior com a instanciação. Isso fica claro ao comparar a Tabela 5 com a Tabela 4. Estas observações indicam que o nível de experiência dos analistas pode influenciar no uso adequado das informações presentes nas receitas. Por exemplo, um analista que nunca ou pouco usou padrões de projeto pode não considerar informações sobre padrões da maneira adequada. Esta observação é importante, para os próximos experimentos realizados com sujeitos humanos. A boa aprovação das informações por parte dos analistas mais experientes também nos motivaram.

Todos os *hot-spots* foram avaliados quanto a utilidade para compreensão do código, seguindo a mesma classificação Likert apresentada anteriormente. A receita para "criar uma nova figura" apresenta informações sobre 24 *hot-spots*, mas somente 2 foram considerados fortemente úteis para a atividade, as classes *hot-spots RectangleFigure* e a *UndoableTool*, sendo que dois sujeitos usaram estes dois *hot-spots* e 3 sujeitos usaram somente a *RectangleFigure* na instanciação. A última pergunta do questionário foi uma questão qualitativa perguntando a percepção geral sobre a utilidade da receita. Todos os participantes relataram que receitas os auxiliou durante o processo de instanciação.

A maioria demonstrou satisfação ao usar a receita, mas sugeriram melhorias na apresentação da informação, justificada pela grande quantidade de informações. Isto levou a melhorias na abordagem, a fim de diminuir este volume e direcionar ainda mais a solução. Na versão usada neste estudo, todos os *hot-spots* presentes no rastro são apresentados na receita. Na versão atual apresentada no Seção 3.2, a receita contempla apenas os *hot-spots* presentes no rastro e ligados às classes específicas que instanciam a característica seguindo os filtros apresentados. Nos próximos experimentos será avaliado se com estes filtros a cobertura se manterá ótima. Além do mais, as receitas passaram a ser estruturadas em um HTML que permite pesquisar e filtrar as informações da receita. Antes, o livro de receitas era estruturado em um documento textual.

4.2.1 Conclusões Preliminares

Os estudos preliminares foram úteis para avaliar se a abordagem é capaz de gerar as receitas com uma cobertura razoável. Esta cobertura é uma das preocupações que ainda serão tratadas nos próximos experimentos. Prova disso é que no segundo estudo preliminar, a taxa de cobertura foi ótima, com 100%. Resultados qualitativos do estudo com sujeitos humanos mostram que a maioria das informações presentes nas receitas foram consideradas úteis em uma atividade real de instanciação de *framework*. O alto volume de informações foi considerado um ponto de melhoria para a abordagem, segundo os sujeitos participantes. Estes resultados permitiram perceber que a abordagem proposta poderá atender as hipóteses levantadas. Mas, melhorias eram necessárias. O que motivou manter as informações da primeira versão do livro de receitas, adicionar novas informações, melhorar os filtros para obter melhor cobertura e diminuir o volume de informações (sendo ainda mais direcionado para a característica desejada) e mudar a apresentação do livro de receitas por meio do HTML que permite a seleção do interesse.

Capítulo 5

Avaliação Experimental dos Livros de Receita

Neste capítulo será apresentado o processo adotado para realizar os experimentos, bem como os resultados e discussões. A Seção 5.1 apresenta o delineamento dos experimentos com sujeitos apresentados neste capítulo. A Seção 5.2 apresenta os resultados dos três experimentos. Na Seção 5.3, os resultados foram discutidos por experimento e por variável de pesquisa. Por fim, na Seção 5.4 é apresentada a conclusão dos resultados obtidos e se estes alcançaram os objetivos de pesquisa.

5.1 Delineamento Experimental

Nesta seção será apresentado o delineamento experimental com informações sobre as perguntas de pesquisa, variáveis, dados coletados, análises estatísticas, seleção dos sujeitos participantes e preparação para execução experimental destes três experimentos.

Três experimentos controlados com sujeitos humanos foram realizados para verificar as hipóteses apresentadas, utilizando a versão final da abordagem de construção de livros de receita. Estes experimentos tiveram como unidade de estudo, o indivíduo durante a execução de atividades de instanciação/desenvolvimento de software utilizando um *framework*. O interesse de estudo foi a comparação destes sujeitos em termos de tempo, taxa de acerto do resultado gerado e percepção de satisfação ao utilizar o livro de receitas proposto e utilizando a documentação tradicional.

Sobre a importância de estudos com sujeitos humanos. VON MAYRHAUSER e LANG (1999) argumentam que muito do que sabemos sobre a compreensão de programas é baseado em estudos observacionais, esta afirmativa é válida até os tempos atuais. Esses estudos geralmente se concentram em entender a abordagem atual do usuário para resolver problemas ou desempenho de tarefas ou procurar compreender o modelo mental do usuário de um conceito, procedimento ou artefato de software. Estudos com sujeitos são mais próximos do cenário real de uso para avaliação de uma nova ferramenta, abordagem

Tabela 6 – Amostra de artigos que apresentam experimentos controlados com sujeitos.

Artigo	Quantidade Sujeitos	Tipo de Atividade	Evento ou Revista
(QUANTE, 2008)	25 graduandos	Questões de Comprensão	ICPC
(MAIA; LAFETá, 2013)	27: 6 graduandos e 21 industria	Implementação de Código	JSS
(HARDER; TIARKS, 2012)	33: 21 graduandos;12 indústria ou academia.	Questões de Compreensão e implementação	ICPC
(CORNELISSEN; ZAIDMAN; DEURSEN, 2011)	34: 28 da academia; 6 da indústria.	Questões de Comprensão	IEEE TRANSACTIONS ON SOFTWARE ENGINEERING
(WETTEL; LANZA; ROBBES, 2011)	41 graduandos	Questões de Comprensão	ICSE
(ROMANO et al., 2016)	47 graduandos	Questões de Compreensão e implementação	ICPC
(HERMANS; AIVALOGLOU, 2016)	61 jovens alunos (idade entre 12 e 14 anos)	8 Questões de Compreensão e 1 de implementação	ICPC
(JAVED; ZDUN, 2014)	109: 92 graduandos;7 da academia.	Questões de Comprensão	WICSA
(FEIGENSPAN et al., 2012)	128 graduandos	Questões de Comprensão	ICPC
(RONG et al., 2015)	135 graduandos	Questões de Comprensão	APSEC

ou documento, isso permite observar resultados mais próximos da realidade. Além do mais, nestes experimentos utilizou-se a aplicação de atividades reais de instanciação com compreensão e alteração de código fonte, com o intuito de observar os resultados em um cenário mais próximo do que ocorre durante a instanciação de *frameworks*.

Experimentos com sujeitos humanos são mais difíceis de serem encontrados, devido a dificuldade na elaboração dos experimentos que envolve muitas variáveis, além da necessidade de contar com a colaboração de terceiros. A maior parte dos experimentos com sujeitos encontrados pela pesquisadora apresentam atividades que envolvem perguntas para serem respondidas, como em (CORNELISSEN; ZAIDMAN; DEURSEN, 2011; QUANTE, 2008). Este número foi menor para trabalhos com atividades reais de desenvolvimento para serem desempenhadas, como se pode ver na amostra apresentada na Tabela 6, esta apresenta alguns exemplos de experimentos controlados. Atividades reais

de desenvolvimento demoram mais tempo para serem realizadas e desgastam mais os participantes. Estas atividades costumam requerer um nível de compreensão mais amplo do problema, software e atividade, fazendo com que menos atividades possam ser exploradas no tempo disponível. E, o desgaste dos participantes pode levar a uma adesão menor e até desistências. Porém, para as hipóteses apresentadas, experimentos com sujeitos e realizando atividades reais se mostraram mais adequados para medir tempo gasto, assertividade e satisfação ao usar as documentações. Seguem as perguntas de pesquisa, que estes experimentos tentaram responder.

5.1.1 Perguntas de pesquisa

Para verificar se os objetivos deste trabalho foram atingidos e verificar a validade das hipóteses, foram definidas algumas perguntas que devem ser respondidas com os dados experimentais:

- Sobre a **cobertura** do livro de receitas:

  P0) *Os livros de receita apresentaram uma cobertura adequada, ou seja, com informações suficientes para guiar a instanciação?*

- Sobre o **tempo de execução** de uma atividade:

  P1) O uso do livro de receitas melhora o tempo de execução de atividades relacionadas à instanciação, ao se comparar com o uso de uma documentação tradicional do *framework*?

- Sobre a **taxa de acerto** na execução de uma atividade:

  P2) *As soluções (código) dadas para uma atividade de instanciação são menos corretas, mais corretas ou igualmente corretas, comparando-se livros de receitas e documentação tradicional?*

- Sobre a **utilidade das informações** fornecidas pela abordagem:

  P3) *As informações fornecidas pela documentação (livro de receitas e tradicional) foram úteis para completar as atividades solicitadas nos experimentos?*

- Sobre o **grau de dificuldade** percebido na execução da **atividade**:

  P4) *Existe diferença entre a percepção do grau de complexidade atribuído à atividade entre os dois tipos de documentação?*

- Sobre o grau de **satisfação** percebido **utilizando a documentação** para execução da atividade:

  P5.1) *Existe diferença entre a facilidade de localização dos interesses de instanciação comparando-se as duas documentações: livros de receita e documentação tradicional?*

P5.2) *Existe diferença entre a clareza das informações comparando-se as duas documentações: livros de receita e documentação tradicional?*

P5.3) *Existe diferença entre o grau de satisfação atribuído ao uso da documentação (livro de receitas e documentação tradicional)?*

5.1.2 Objetos

Dois *frameworks* Java são objetos desta pesquisa. A abordagem apresentada foi desenvolvida para *frameworks* Java, o que facilita a seleção de objetos e sujeitos, pois existe um grande número de *frameworks* nesta linguagem e esta é uma linguagem amplamente ensinadas nas universidades e utilizada no mercado.

A Tabela 7 apresenta algumas métricas destes dois *Frameworks*: JHotDraw 5.3 e JFace 3.4.2. Estes apresentam tamanhos e aplicabilidade distintas. O JFace é significativamente maior do que o JHotdraw. JFace é um *framework* para criar interfaces para plug-ins Eclipse de uso comercial. Enquanto o JHotDraw é um *framework* para criar sistemas para desenho de diagramas. Esta diferença contribui para verificar se a abordagem é aplicável com boa cobertura e desempenho para *framewoks* com diferentes aplicações e tamanhos. Wettel WETTEL et al. (2011) apresentam um survey, onde realizaram o levantamento exaustivo de trabalhos de pesquisa com validação experimental na engenharia de software, e neste artigo eles listaram alguns desejos para o delineamento experimental, um deles é incluir mais de um sistema no desenho experimental. QUANTE (2008) mostrou que realizar a mesma experiência em dois sistemas diferentes pode levar a resultados significativamente diferentes. Além de escolher sistemas do mundo real. Alguns experimentos na literatura utilizam pequenos sistemas, dificultando a generalização resultados para sistemas do mundo real.

Tabela 7 – Métricas dos *Frameworks* JHotDraw e JFace, objetos de experimento.

Framework	Qtd. Classes	Qtd.Métodos	Pacotes
JHotDraw 5.3	211	1924	11
JFace 3.4.2	271	4580	17

5.1.2.1 JHotDraw

O *framework* JHotDraw, versão 5.3, já foi apresentado anteriormente. Os fatores que levaram a seleção deste *framework* para os experimentos são:

- O *framework* é utilizado em pesquisas ciêntificas e ter sido utilizado anteriormente pelos pesquisados em uma pesquisa que envolvia análise dinâmica. Logo, já era conhecido que este seria aderente ao uso da análise dinâmica;

- Apresenta boa documentação e controle de versões;
- Facilidade em encontrar exemplos executáveis de sistemas que utilizam o JHotdraw, no seu próprio site encontramos os dois sistemas utilizados para gerar as receitas;
- O código fonte do *framework* é disponibilizado. Alguns *frameworks* apenas disponibilizam os arquivos .jar. Esta é uma necessidade da abordagem;
- O domínio de aplicação do *framework* é de fácil compreensão para os potenciais sujeitos participantes dos experimentos.

Para gerar o livro de receitas do *framework* JHotDraw foram trabalhadas 31 características, estas foram obtidas dos exemplos presentes nos sistemas JModeller e JHotDraw App, que utilizam este *framework*. O sistema JModeller é uma ferrameta para criar diagramas de classe. O sistema JHotDraw App possibilita desenhar imagens com figuras diversas da geometria, com textos e conectores. Ambos os sistemas estão disponíveis no site oficial do *framework*[1]. Para estas características, foram obtidas 41 classes e 48 métodos *hot-spots* distintos que compuseram o livro de receitas com 292 páginas HTML. Outro ponto importante era que o livro de receitas deve ser volumoso como o documento tradicional utilizado. A avaliação de cobertura para esta 31 caracteríticas foram úteis na detecção de melhorias na ferramenta CookFrame. Este livro de receitas apresentou uma taxa de cobertura de 100% das classes e 100% dos métodos envolvidos na instanciação das características para a nova versão da abordagem. Para medir esta cobertura, um aluno de iniciação científica realizou a análise do código dos exemplos para listar as classes, métodos e como estes foram utilizados para as 31 características selecionadas. Logo depois da análise manual, ele foi apresentado ao Livro de Receita para comparar os resultados deste com a análise manual e obter a cobertura dos dados.

A documentação tradicional do JHotDraw utilizada está estruturada como um HTML e apresenta:

- *i)* diagrama de classes simplificado, com as 8 classes principais do *framework*;
- *ii)* características que se pode instanciar com o *framework*;
- *iii)* organização dos pacotes, com referência para o Javadoc;
- *iv)* visão geral dos sistemas exemplo entregues pelo JHotDraw para auxiliar na instanciação, disponíveis no código do *framework* para os usuários;
- *v)* instruções sobre como compilar o *framework*;
- *vi)* informações sobre as versões do *framework*;
- *vii)* Javadoc com todos os pacotes, classes e métodos do *framework* destalhados.

[1] http://www.jhotdraw.org/

5.1.2.2 JFace

O segundo *framework* é o JFace[2], versão 3.4.2, com o SWT[3] utilizados para criar plug-ins do Eclipse. JFace é um toolkit com classes para desenvolver diversas tarefas comuns de programação ligadas à interface do usuário. O JFace inclui os componentes usuais de imagem, registros de fontes, estruturas de texto, diálogo, preferências e assistente de relatórios, para criar visões que simplificam a apresentação de dados de aplicação estruturados como listas, tabelas ou árvores. Os fatores que levaram a seleção deste *framework* para os experimentos são:

- *Framework* utilizado para o desenvolvimento de *plug-ins* Eclipse. Portanto, existe uma variedade de exemplos de plug-ins Eclipse que utilizam o JFace. Na plataforma de suporte ao desenvolvedor Eclipse existem exemplos para download;
- Este já foi utilizado em trabalhos relacionados como (HOLMES; MURPHY, 2005), (HEYDARNOORI et al., 2012);
- Apresenta boa documentação online[4] e controle de versões;
- O código fonte do *framework* é disponibilizado conforme a versão, junto aos exemplos de uso do mesmo;
- O domínio de aplicação do *framework* é de fácil compreensão para os potenciais sujeitos.

Para gerar o livro de receitas do *framework* JFace e SWT foram trabalhadas 16 características ligadas as Visões que podem ser desenvolvidas com este *framework*. Estas foram obtidas dos exemplos de aplicações disponibilizados na plataforma para desenvolvedores Eclipse. Para estas características, foram obtidas 66 classes e 120 métodos *hot-spots* distintos que compuseram o livro de receitas com 250 páginas HTLM. Para este Livro de Receitas, a taxa de cobertura média foi de 100% das classes e 95,88% dos métodos envolvidos na instanciação das características. Os métodos faltantes não foram identificados no rastro. Apenas uma característica apresentou dois falsos positivos, caracterizando 0,5% de métodos falsos positivos para esta característica. Para obter a taxa de cobertura, o mesmo processo de análise manual realizado com o JHotDraw foi realizado para o livro de receitas do JFace e SWT, porém realizado por um desenvolvedor JAVA experiente.

A documentação tradicional do JFace e SWT disponível na página oficial do JFace Eclipse[5] é completa e bem explicada, mas traz mais informação do que o necessária para a atividade simples do experimento. Então, selecionou-se a documentação sobre o JFace e SWT disponível no Guia de Desenvolvedores de Plugins do Eclipse[6]. Por ser uma

[2] https://wiki.eclipse.org/JFace
[3] https://www.eclipse.org/swt/
[4] http://help.eclipse.org
[5] https://wiki.eclipse.org/JFace
[6] http://help.eclipse.org

documentação mais direta e possuir um tópico específico para as visões que podem ser desenvolvidas e são o interesse direto das atividades aplicadas. Durante as atividades, os desenvolvedores foram instruídos a utilizar apenas este tópico da documentação. Como a documentação está completamente em inglês para os dois **frameworks**, os alunos foram instruídos a usar uma ferramenta de tradução se preferissem o documento em inglês, ou usarem o documento traduzido pelos pesquisadores. Esta documentação apresenta:

- *i)* Uma introdução à construção de visões utilizando o JFace, com as possíveis visões que se pode implementar;
- *ii)* Listas com explicação sobre classes *hot-spot* que podem ser utilizadas para desenvolver as Views, que inclui as duas Views solicitadas a serem implementadas para as atividades;
- *iii)* Os nomes das classes nesta lista são um link para um Javadoc com informações sobre estas classes, pacotes, seus métodos e parâmetros;
- *iv)* Ao longo do documento, vários exemplos de código são apresentados para o desenvolvimento das visões, inclusive exemplos úteis para as duas atividades definidas para o experimento.

5.1.3 Desenho dos Experimentos

Três experimentos foram definidos com o objetivo de responder às perguntas de pesquisa. Estes irão se diferenciar pelo nível de conhecimento dos participantes, *framework* utilizado, sistemas objeto e complexidade das atividades aplicadas por experimento. Conforme trabalhos relacionados que realizaram experimentos com sujeitos humanos, mostrados na Tabela 6, definiu-se um número de 15 a 20 participantes por experimento, como um número minimamente adequado. Diante do número de participantes que aceitaram participar por instituição e os profissionais de mercado, foi possível elaborar 3 experimentos. Um experimento exclusivamente com profissionais e dois experimentos com alunos, separados por instituição. Por este motivo, a quantidade de sujeitos variou conforme o experimento. Esta divisão facilita a aplicação dos experimentos, pois são instituições em estados diferentes e profissionais que moram e diferentes lugares e trabalham em diferentes empresas.

Cada experimento contou com 2 grupos de sujeitos, conforme a Tabela 8. Cada sujeito realizou duas atividades, hora usando a documentação tradicional do *framework* (grupo Controle), hora usando o livro de receitas (grupo Receitas). A alternância dos sujeitos com o uso da documentação na execução das duas atividades, visa mitigar a ameaça ao experimento causada pelos diferentes níveis de conhecimento e interesse dos sujeitos.

Tabela 8 – Organização dos Grupos de Sujeitos, Atividades, *Framework* e Documentação

Experimento	Grupo de Sujeitos	Qtd. de sujeitos	Atividade	*Framework*	Grupo Experimental
Instituição 1	Alunos A	7	Atividade 1.1	JFace	Controle
Instituição 1	Alunos B	7	Atividade 1.1	JFace	Receitas
Instituição 1	Alunos A	7	Atividade 1.2	JFace	Receitas
Instituição 1	Alunos B	7	Atividade 1.2	JFace	Controle
Instituição 2	Alunos C	8	Atividade 2.1	JHotDraw	Controle
Instituição 2	Alunos D	7	Atividade 2.1	JHotDraw	Receitas
Instituição 2	Alunos C	8	Atividade 2.2	JHotDraw	Receitas
Instituição 2	Alunos D	7	Atividade 2.2	JHotDraw	Controle
Profissionais	Profissionais A	8	Atividade 3.1	JHotDraw	Controle
Profissionais	Profissionais B	7	Atividade 3.1	JHotDraw	Receitas
Profissionais	Profissionais A	8	Atividade 3.2	JHotDraw	Receitas
Profissionais	Profissionais B	7	Atividade 3.2	JHotDraw	Controle

5.1.3.1 Atividades

Duas atividades reais de instanciação foram definidas para cada experimento. Em cada atividade proposta, definiu-se a necessidade de implementar os relacionamentos entre o código do usuário e o *framework* (*hot-spots* - API do *framework*). As atividades foram definidas procurando atender ao requisito de estarem próximas em escopo e complexidade a tarefas reais realizadas por profissionais (WETTEL; LANZA; ROBBES, 2011).Foram definidas atividades que possuíam exemplos e explicações tanto no Livro de Receitas como na documentação Tradicional que pudesse ajudar na atividade. Estas atividades não eram exatamente iguais às soluções exemplos explicadas no livro de receitas. Por exemplo, uma atividade era a criação de uma nova figura Hexágono para uma ferramenta que cria diagramas, o Livro de Receitas trazia exemplos de como criar uma nova figura para desenhar losangos, círculos e quadrados.

Considerando a alternância dos grupos entre Grupo Controle e Grupo Receitas, deve-se considerar que o primeiro contato com o *framework* pode eventualmente influenciar no segundo contato. É importante minimizar esta influência, e por isto, procurou-se definir atividades que explorassem *hot-spots* e trechos de códigos diferentes entre as atividades. Além disso, as atividades deveriam ter o mesmo nível de complexidade e serem similares em tempo de execução, para poder comparar os resultados de cada sujeito, tendo como única variável o tipo de documentação utilizada. O nível de complexidade foi atribuído para uma possível solução testada, conforme o número de *hot-spots* necessários, a quantidade de classes e métodos a serem criados, o número aproximado de linhas da solução esperada e o tempo gasto com a atividade. Algumas atividades foram definidas e testadas com um desenvolvedor que não participou dos experimentos. Nestes testes, o tempo e a complexidade das atividades foram medidos e considerados similares.

Para os grupos de alunos, as atividades de instanciação foram elaboradas de modo serem possíveis de se realizar em um tempo limitado de 50 minutos. A limitação do tempo permitido para resolver cada atividade evita que os participantes gastem todo o tempo reservado para o experimento na resolução de uma única tarefa, como a experiência de QUANTE (2008) conforme citado em (WETTEL; LANZA; ROBBES, 2011). Definiu-se um espaço de tempo próximo a dois horários de aula para poder utilizar os horários dos alunos em sala e ter uma boa adesão dos participantes. Para as duas atividades, totalizou-se 1 hora e 40 minutos de atividade, mais 30 minutos de treinamento. Para os profissionais, definiu-se atividades mais complexas e possíveis de se realizar em um intervalo de duas horas, não limitando o tempo de execução da atividade para estes. No convite para participação dos experimentos, três horas destes profissionais foram solicitadas para cada atividade. Além disso, solicitou-se que as duas atividades fossem realizadas em dias distintos para não "desgastar" os sujeitos profissionais. Solicitou-se que os participantes anotassem as pausas realizadas para que este tempo fosso subtraído do tempo total gasto com a atividade.

A seguir são descritas as atividades realizadas por experimento:

❑ **Experimento 1 – Alunos Instituição 1 – *Framework* JFace**

Neste experimento, uma classe Java incompleta foi passada aos alunos, para cada atividade. Eles foram instruídos a completar estas classes utilizando o framework JFace para executar as atividades.

- **Atividade 1.1:** Criar uma tabela simples com valores de 0 à 9 em cada linha da primeira coluna. A implementação da inserção de valores nas linhas estava pronta na classe Java passada aos alunos. **Complexidade:** Uma possível solução seria completar a classe dada com dois métodos faltantes, e usar o *framework* para definir a estrutura tabela. Um dos métodos é o construtor da classe e o outro é a main( ). No construtor, 7 *hot-spots* são utilizados (3 instanciações de objetos e 4 invocações de métodos). Na main( ), 9 *hot-spots* são utilizados (3 instanciações de objetos e 6 invocações de métodos).
- **Atividade 1.2:** Criar uma árvore simples com valores de 0 à 9 em cada nó e acrescentar os nós internos com valores que vão até o número do nó. A implementação para definir a inserção de valores e quantidade de nós internos estava pronta na classe passada aos alunos. **Complexidade:** Uma possível solução seria completar a classe dada com dois métodos faltantes, e usar o *framework* para definir a estrutura árvore. Um dos métodos é o construtor da classe e o outro é a main(). No construtor, 6 *hot-spots* são utilizados (3 instanciações de objetos e 3 invocações de métodos). Na main(), 9 *hot-spots* são utilizados (3 instanciações de objetos e 6 invocações de métodos).

❑ **Experimento 2 – Alunos da Instituição 2 – *Framework* JHotDraw**

Neste experimento, solicitou-se aos alunos a realização de melhorias no sistema JModeller, com a inclusão de novas funcionalidades. O JModeller é um sistema que desenha diagramas de classe e utiliza o *framework* JHotdraw.

- **Atividade 2.1 :** No menu para colorir figura, criar uma opção que muda a cor da figura Classe selecionada, qualquer outra cor. **Complexidade:** Uma possível solução seria alterar o método do menu já existente, inserindo um novo item. Já existem itens neste menu. Este novo item do menu irá invocar um novo método (criado nesta mesma classe) que invoca um *hot-spots* que permite colorir a figura na cor desejada por meio da passagem de parâmetro. .
- **Atividade 2.2 :** No menu para mover figura, criar uma opção que move a figura Classe selecionada para uma posição na imagem que seja mais ao centro, qualquer cordenada. **Complexidade:** Uma possível solução seria alterar o método do menu já existente, inserindo um novo item. Já existem itens neste menu. Este novo item do menu irá invocar um novo método (criado nesta mesma classe) que invoca um *hot-spot* que permite mover a figura para a coordenada desejada, passadas por parâmetro.

❑ **Experimento 3 – Profissionais – *Framework* JHotDraw**

Neste experimento, solicitou-se aos profissionais a realização de melhorias no sistema JModeller, com a inclusão de novas funcionalidades.

- **Atividade 3.1:** Criar uma nova figura que não existe no sistema e nem no *framework* para desenhar hexágonos, e incluir este item no menu de figuras. **Complexidade:** Uma possível solução seria criar uma nova classe para a figura Hexagon (exemplo, HexagonFigure). Na classe principal do sistema, invocar esta classe criada para adicionar este novo item ao menu de figuras. Esta nova classe deve estender um *hot-spot*. Espera-se que aproximadamente oito métodos sejam criados nesta nova classe, onde quatro sejam redefinição de métodos. Espera-se também a invocação de cinco métodos *hot-spots* e a instanciação de três classes *hot-spots*. Totaliza-se 17 *hot-spots* envolvidos.
- **Atividade 3.2:** Criar um novo conector de figuras que não existe no sistema e nem no *framework* para desenhar um conector que apresenta um pentágono na extremidade fim, e incluir este item no menu de figuras. **Complexidade:** Uma possível solução seria criar uma nova classe para o novo conector (ex. NewConnector). E, na classe principal do sistema, invocar esta classe criada para adicionar este novo item ao menu de figuras. Esta nova classe deve estender um *hot-spots*. Espera-se que aproximadamente quatro métodos sejam

criados nesta nova classe, onde três sejam redefinição de métodos. Espera-se também a invocação de doze métodos *hot-spots* e a instanciação de quatro classes *hot-spots*. Totaliza-se 21 *hot-spots* envolvidos.

5.1.4 Sujeitos

Todos os sujeitos participantes são voluntários não envolvidos previamente com a pesquisa. A escolha pela participação de alunos e profissionais de áreas diversas do mercado profissional e acadêmico, parcialmente mitiga a preocupação de Di Penta et al., que argumenta que "um grupo composto apenas de estudantes pode não representar adequadamente a população de possíveis usuários" (PENTA; STIREWALT; KRAEMER, 2007).

Foram identificadas cinco questões sobre estes sujeitos que poderiam influenciar nos resultados, sendo elas: *i)* nível de conhecimento Java; *ii)* nível de conhecimento sobre o uso do IDE Eclipse, utilizado durante a execução das atividade; *iii)* nível de conhecimento sobre o uso de *frameworks*; *iv)* quantidade de sistemas já desenvolvidos, somente para os alunos; e, *v)* tempo de experiência, somente para os sujeitos profissionais de mercado. Um questionário de seleção contempla peguntas sobre estas questões (veja o Anexo 2) que auxiliaram na caracterização dos sujeitos participantes, seleção e distribuição dos sujeitos nos grupos, a fim de compor os grupos de forma mais similar possível, por meio de uma randomização estratificada. A seguir serão apresentadas informações sobre o processo de seleção da amostra de sujeitos participantes.

Alunos: Para seleção da amostra de alunos, professores das disciplinas que contemplam as disciplinas de interesse para a pesquisa (Java, *frameworks* e padrões de projeto) das seguintes instituições foram contactadas e autorizaram a realização do contato com seus alunos e experimentos em suas acomodações. Instituições:

- UFU - Universidade Federal de Uberlândia – Faculdade de Computação – Uberlândia/MG;
- FEMASS - Faculdade Professor Miguel Ângelo da Silva Santos – Macaé/RJ.

Na apresentação dos resultados, não serão identificados quais são os grupos de alunos da UFU ou FEMASS, iremos chamá-los de Instituição 1 e 2. Essa medida é para preservar os participantes e instituições. Os professores contactados viabilizaram o contato com os alunos que foram convidados à participar. Estes receberam um questionário online a fim a avaliar quais se enquadram nos critérios de inclusão e exclusão destes experimentos e se voluntariarem a participar. A adesão foi de 14 alunos da Instituição 1 e 15 alunos da Instituição 2.

Profissionais: Para seleção da amostra de profissionais, foram contactados aproximadamente 50 analistas de mercado (possíveis sujeitos participantes), que fazem parte da rede de amigos e colegas dos pesquisadores. Estes receberam um questionário online a

fim a avaliar quais se enquadram nos critérios de inclusão e exclusão destes experimentos e se voluntariarem a participar.

Conseguiu-se a participação efetiva de 15 profissionais, 9 da academia e 6 da industria. Alguns dos sujeitos da academia tiveram passagem pela indústria. Os sujeitos serem contato dos pesquisadores poderia caracterizar um viés para a pesquisa. Entretanto, este viés é bastante mitigado, pois as perguntas principais de pesquisa não são subjetivas e envolvem uma atividade real de instanciação de um *framework*. Além do mais, todos os sujeitos utilizaram as duas documentações e não foram informados sobre qual documentação era da abordagem proposta.

Conforme se pode ver na Tabela 6, que apresenta uma amostra dos trabalhos que apresentam experimentos controlados com sujeitos encontrados na literatura, o volume de 44 sujeitos participantes, 88 execuções experimentais com sujeitos, é um valor compatível com o apresentado pela literatura para trabalhos recentes apresentados em eventos ou periódicos relevantes. Como se pode se observar na amostra, a maioria dos trabalhos encontrados contou com a participação de alunos, por ser mais fácil obter a adesão deste perfil de sujeitos no meio acadêmico. E, conforme já argumentado anteriormente, a maioria dos trabalhos encontrados não realizam atividades com alteração do código fonte, implementação.

5.1.4.1 Critérios de inclusão

A seguir serão apresentados os critérios de inclusão dos sujeitos voluntários, que permite que estes participem da pesquisa:

Alunos: Para os experimentos com alunos, foram convidados alunos que atendessem aos seguintes critérios de elegibilidade:

- ❑ Graduandos dos cursos de ciência da computação, sistema de informação e/ou engenharia da computação;
- ❑ Estarem cursando a disciplina que contempla a linguagem e programação JAVA.
- ❑ Estarem no final do semestre e terem realizado avaliações na disciplina.

Profissionais: Para o experimento com profissionais, analistas desenvolvedores foram selecionados seguindo os seguintes critérios de elegibilidade:

- ❑ Formação acadêmica em ciência da computação, sistemas de informação ou engenharia da computação.
- ❑ Desenvolvedores com experiência de programação na linguagem JAVA;
- ❑ Desenvolvedores que já tenham utilizado *frameworks*;

Estes critérios levaram à exclusão de alguns possíveis participantes, antes mesmo de responderem aos questionários.

5.1.4.2 Características da Amostra

Como o questionário de seleção foi possível traçar um perfil dos sujeitos voluntários para realizar a seleção e distribuição nos grupos, a seguir estes grupos serão caracterizados conforme as questões de seleção.

Sobre os grupos de **Alunos** - Instituição 1:

- Todos são alunos dos cursos de Ciência da Computação ou Sistema de Informação da Universidade Federal de Uberlândia e já estavam ou já haviam cursado a disciplina sobre desenvolvimento orientado à objetos;
- Sobre p nível de conhecimento sobre orientação à objetos e java atribuído pelos próprios sujeitos, temos que que numa escala likert de 1 à 5, 63,34% atribuiu o nível 3, 27,26% atribuiu o nível 4 e 9,09% atribuiu o nível 5.
- Este grupo apresentou uma média de 4,8 aplicações criadas em java e orientadas à objetos por sujeito.
- Mais da metade, 54,55%, já trabalhou com desenvolvimento de software por menos de 1 ano. Apenas um aluno tem experiência superior à 2 anos;
- Sobre o IDE usado, 54,55% dos alunos já haviam tido contato com o IDE Eclipse utilizado durante a atividade;
- Sobre o nível de conhecimento com o uso de *frameworks*, em uma escala Likert de 1 a 5: 27,27% dos alunos atribuiu o nível 3; 54,55% atribuiu o nível 2; e 18,18% atribuiu o nível 1.

Sobre os grupos de **Alunos** - Instituição 2:

- Todos são alunos do curso de Sistema de Informação da Faculdade Professor Miguel Ângelo da Silva Santos e estavam ou já haviam cursado a disciplina sobre desenvolvimento orientado à objetos;
- Sobre p nível de conhecimento sobre orientação à objetos e java atribuído pelos próprios sujeitos, temos que que numa escala likert de 1 à 5, 20% atribuiu o nível 1, 26,67% atribuiu o nível 2, 46,67% atribuiu o nível 3 e 6,67% atribuiu o nível 5.
- Este grupo apresentou uma média de 2,5 aplicações criadas em java e orientadas à objetos por sujeito.
- Apenas 26,67% já trabalhou com desenvolvimento de software. Dois alunos possuem a experiência de 1 ano; um aluno possui a experiência de 2 anos; e um aluno possui a experiência de 3 anos com desenvolvimento de software;

- Sobre o IDE Eclipse, 33,33% dos alunos já haviam tido contato com o IDE utilizado durante o experimento;
- Sobre o nível de conhecimento sobre o uso de *frameworks*, em uma escala Likert de 1 a 5: 33,33% dos alunos atribuiu o nível 3; 33,33% atribuiu o nível 2 e 33,33% atribuiu o nível 1.

Sobre os sujeitos participantes **Profissionais A e B**:

- Todos possuem formação acadêmica em Ciência da Computação ou Sistemas de Informação. Sendo 13% especialistas; 40% mestres; e 6% de doutorandos.
- O tempo de experiência profissional variou, sendo que 33,33% dos sujeitos apresentam de 4 à 6 anos de experiência e 40% sujeitos está na faixa de 10 à 13 anos de experiência. Sobre a área de atuação, 73,33% foram ou são desenvolvedores JAVA.
- Sobre o nível de conhecimento em orientação à objetos, em uma escala Likert de 1 a 5 sobre o nível de conhecimento: 13,33% dos sujeitos considerou como nível 3; 60% considerou com 4; e 26,67% considerou como 5.
- Sobre o nível de conhecimento relativo ao uso de *frameworks*, em uma escala Likert de 1 a 5: 46,66% dos sujeitos marcou o nível 3, 46,66% marcou o nível 4 e 0,6% marcou nível 5.
- Sobre o conhecimento de padrões de projeto, em uma escala Likert de 1 a 5: 73,33% dos sujeitos marcou nível 3 e 13,33% marcou nível 4.

Os grupos de alunos e profissionais apresentam uma boa variabilidade de sujeitos. O grupo de profissionais apresenta sujeitos com uma boa experiência em desenvolvimento, considerada suficiente para realizar as atividades do experimento. Wettel e colegas em (WETTEL; LANZA; ROBBES, 2011) falam da importância de envolver sujeitos participantes da indústria. Qualquer abordagem concebida para apoiar os profissionais em seu trabalho é melhor avaliada usando uma amostra de sujeitos com uma parcela justa de profissionais de desenvolvimento de software. A amostra de alunos Instituição 1 e Instituição 2 é caracterizada por uma maioria de alunos com pouca experiência com desenvolvimento e uso de *frameworks*. Wettel e colegas reforçam a importância de tomar em consideração o nível de experiência dos participantes. A experiência pode influenciar no resultado do experimento (WETTEL; LANZA; ROBBES, 2011).

5.1.5 Procedimento Experimental

Esta seção descreve os procedimentos realizados durante os experimentos. Estes variaram entre os experimentos com alunos e profissionais. Os dois experimentos realizados com alunos seguiram os mesmos procedimentos e foram realizados em sala de aula com

a presença da pesquisadora. O experimento com profissionais seguiram procedimentos parecidos, porém foi realizado remotamente e o tempo para realizar as atividades não foi limitado. Espera-se um maior controle sobre os experimentos realizados em um ambiente com a presença do pesquisador, que permite verificar a execução correta do processo e tirar dúvidas dos participantes. Os experimentos remotos requerem cuidados para ter este mesmo controle. Entretanto, a flexibilidade do experimento remoto auxiliou na adesão dos sujeitos profissionais e no processo de aplicação do experimento, dado que os participantes trabalham em empresas, cidades, e estados distintos.

A seguir serão apresentadas as etapas dos procedimentos experimentais.

5.1.5.1 Preparação do Ambiente

Os *frameworks* necessitam de IDEs e plataformas específicas instaladas para serem utilizados. Estes foram instalados e testadas nos laboratórios da Instituição 1 e Instituição 2 antes da etapa experimental com os alunos. Testar corretamente todas as maquinas é fundamental para não haver influência deste fator nos resultados.

Para o experimento com os profissionais, os sujeitos receberam os arquivos com as ferramentas necessárias e um documento de instrução para preparar o ambiente que testar antes da execução experimental. O procedimento é tranquilo e não apresentou problemas.

Não foi definida uma configuração mínima das máquinas para os experimentos, pois, o software e *framework* com o IDE Eclipse das atividades rodam com facilidade em qualquer máquina de desenvolvedor ou máquinas de laboratórios para o desenvolvimento de software.

5.1.5.2 Instrução e Treinamento dos sujeitos

Antes da execução das atividades, todos os sujeitos participantes (alunos e profissionais) foram instruídos sobre o procedimento experimental. Documentos com os passos e cuidados na execução dos experimentos foram criados para os alunos e profissionais. Para os profissionais, estes documentos foram mais detalhados, pois eles executaram os experimentos remotamente. Para estes, o passo a passo de execução foi passado por e-mail com todos os arquivos necessários e explicação dos procedimentos e cuidados necessários antes de realizar a atividade. Foram enviados dois e-mails, um para cada atividade. Para os alunos, um email foi elaborado e enviado com algumas instruções importantes para o experimento em sala de aula, local, horário, tempo necessário, e uma descrição sobre como seria o procedimento. Em sala de aula, as instruções sobre o procedimento de execução foram reforçadas e complementada.

Depois das instruções inciais, os participantes foram apresentados às estruturas do livro de receitas e da documentação tradicional (controle) para o melhor uso destes. Os participantes não foram apresentados aos conteúdos neste treinamento, apenas ao conceito, organização e ambientes de instanciação. Este treinamento teve duração média de

30 minutos, aproximadamente 15 minutos para cada documentação, tanto para os experimentos com profissionais como para os alunos. Para os experimentos com profissionais de mercado, criamos dois vídeos para explicar os documentos livro de receita e o documento tradicional, também com duração média de 15 minutos para cada documento. Os profissionais foram instruídos a assistirem os vídeos de treinamento antes da realização das atividades.

Os sujeitos foram instruídos igualmente via documentação e apoio do pesquisador sobre as atividades a serem desempenhadas visando diminuir os desvios, falhas e dúvidas durante a execução das atividades. As atividades eram simples e foram descritas de forma textual em um documento com figuras que mostravam a modificação desejada.

Os treinamentos e materiais de instrução constituem uma etapa importante dos experimentos, pois garantem o mínimo de conhecimento necessário aos sujeitos participantes. Este treinamento, se bem executado, minimiza desvios que podem ocorrer pela falta de conhecimento sobre o uso das documentações, ambiente utilizado e processo da execução. Os participantes remotos foram contactados ao longo de todo processo para se ter certeza que houve uma boa compreensão e execução do procedimento.

5.1.5.3 Execução experimental

Após a seleção e instrução dos sujeitos foram realizados os experimentos. Para os alunos da Instituição 1 e Instituição 2 foram definidos dias e horários distintos para obter maior adesão por meio da flexibilidade. Os profissionais realizaram as atividades no dia e horário conveniente a este, dentro de um prazo de entrega da atividade. A diferença entre a execução realizada com o grupo Controle e o grupo Receitas para um experimento foi a documentação utilizada, variável independente deste experimento. É importante fornecer os mesmos dados a todos os sujeitos participantes para que o efeito observado do experimento seja adequadamente atribuído para as variáveis independentes (WETTEL; LANZA; ROBBES, 2011).

No procedimento executado com os alunos, estes foram instruídos previamente por e-mail. Ao chegarem todos os alunos agendados para o dia e horário, o experimento iniciou. Todas as máquinas já estavam configuradas o IDE com o sistema ou classe a ser alterada, a documentação a ser utilizada e questionários. Eles foram novamente instruídos, separados por grupos e foram apresentados às documentações. Cada grupo foi alternadamente instruído a utilizar uma das documentações. Foram instruídos a executarem a primeira atividade utilizando a documentação em um tempo máximo de 50 minutos, onde aqueles que terminassem a atividade antes deveriam marcar o horário de término e então salvar as alterações feitas em código e preencher o questionário relativo àquela atividade e ao grupo que ele faz parte. Após os 50 minutos, aqueles que não terminaram, foram instruídos a parar a atividade, salvar o código alterado e preencher o questionário. Foi dado um tempo de aproximadamente 15 minutos para preencher os questionários. Este foi considerado um

tempo suficiente para preencher o mesmo. Entretanto, alguns sujeitos demandaram mais tempo, e este tempo foi concedido. Após o preenchimento do questionário, eles foram apresentados à nova atividade que seguiu o mesmo procedimento. Após a execução do experimento, todos os códigos foram coletados e associados aos seus executores e grupos.

No procedimento executado com os profissionais, o e-mail continha todas as informações necessárias para que o experimento fosse executado. Neste, eles foram instruídos à: *i)* baixar todos os arquivos e documentos, verificar se todos estão íntegros e realizarem as instalações, em caso de problemas contactar a pesquisadora; *ii)* assistir o vídeo com o treinamento sobre a documentação atribuída à ele para aquela atividade; *iii)* executar a atividade em um dia que pudessem se dedicar a esta; *iv)* que coletassem o tempo de início, fim e possíveis pausas (exemplo: pausa para atender a um telefone), e que não contemplassem o tempo de treinamento e instrução neste tempo coletado; *v)* utilizar a documentação e observar a sua utilidade durante a atividade; *vi)* salvar o código alterado por eles e enviar por email para a pesquisadora ao final da atividade; e, *vii)* preencher o questionário enviado para a atividade, logo após a execução da atividade. Eles também foram informados que a atividade não teria tempo limite, mas que eles poderiam desistir a qualquer momento caso eles não conseguissem concluir a mesma, ou concluir parcialmente. Ao longo do processo a pesquisadora se colocou a disposição para responder dúvidas e dar apoio remotamente.

Mesmo aqueles que desistissem foram instruídos a responder o questionário. Após o envio do questionário e código da primeira atividade para a pesquisadora, os sujeitos receberam a segunda atividade que seguiu o mesmo procedimento, porém para ser executada com outra documentação.

O questionário para coleta dos dados foi preenchido por todos os participantes após a execução de cada atividade, ou seja, cada participante respondeu dois questionários. Os questionários de coleta dos dados, conforme o grupo, foram similares para os 3 experimentos. A Tabela 9 apresenta as perguntas de pesquisa e algumas das perguntas dos questionários relacionadas. Todas as perguntas foram iguais para os dois grupos (Controle e Receitas), exceto a pergunta P3. A pergunta P3 teve o mesmo sentido para ambos os grupos, e somente foi escrita com algumas diferenças devido à estrutura do documento a ser avaliado.

5.1.6 Variáveis

Todos os experimentos são limitados pelas mesmas variáveis independentes, dependentes e não controladas, que serão apresentadas a seguir. Para responder as perguntas P1 à P5, a **variável independente** é:

O uso do livro de receitas ou documentação tradicional, variando conforme o grupo. Os sujeitos participantes serão distribuídos em dois diferentes grupos: sendo que cada grupo hora atuará como **Grupo Controle** (que não usa a abordagem) e hora como um

Tabela 9 – Perguntas de pesquisa e algumas das perguntas dos questionários relacionadas.

Pergunta de Pesquisa	Pergunta do Questionário
P1	**Controle e Receitas:** Digite abaixo o horário de Início e Fim de execução da Atividade.
P2	**Controle e Receitas:** Você concluiu a atividade com sucesso?
P3	**Controle:** Sobre as informações apresentadas na Documentação, qual foi a utilidade destas para a execução da Atividade? **Receitas:** Sobre os *hot-spots* do *framework* sugeridos na(s) Receita(s), as informações sobre estes foram uteis para a realização da tarefa?
P4	**Controle e Receitas:** Esta atividade foi COMPLEXA de realizar?
P5.1	**Controle e Receitas:** Foi FÁCIL LOCALIZAR as informações necessárias para instanciação no Documento? O mecanismo de navegação pelo documento foi útil para a atividade?
P5.2	**Controle e Receitas:** As informações dispostas no documento estavam CLARAS, ou seja, facilmente compreensíveis e bem apresentadas?
P5.3	**Controle e Receitas:** Você ficou SATISFEITO com o uso do Documento para realizar esta atividade?

Grupo Receitas (usa o livro de receitas proposto). O uso e não uso das informações presentes livro de receitas permitirá comparar o desempenho da abordagem proposta.

As **variáveis dependentes** são:

Para a pergunta **P1** a variável dependente é:

- O tempo necessário para executar a atividade de manutenção nos dois grupos e por atividade. Foi programada a coleta dos horários (horas e minutos) do início e fim das atividades de cada sujeito.

Para a pergunta **P2** a variável dependente é:

- A taxa de acerto das soluções nos dois grupos e por atividade, conforme o número esperado de relacionamentos corretamente estabelecidos com os *hot-spots* do *framework*. Além disso, programou-se a execução de casos de teste para verificar a cobertura da solução para o que foi requisitado.

Para a pergunta **P3** a variável dependente é:

- Resposta dos sujeitos dos grupos Controle e Receitas em relação à utilidade das informações fornecidas nas documentações (Livro de Receitas ou tradicional).

Para a pergunta **P4** a variável dependente é:

- Resposta dos sujeitos dos grupos Controle e Receitas em relação à percepção da dificuldade na execução das atividades.

Para responder a pergunta **P5** a variável dependente é:

- Resposta dos sujeitos dos grupos Controle e Receitas em relação à facilidade de localização dos interesses de instanciação nas documentações para execução das atividades
- Resposta dos sujeitos dos Grupos Controle e Receitas em relação à clareza das informações nas documentações para execução das atividades
- Resposta dos sujeitos dos grupos Controle e Receitas em relação à satisfação com o uso das documentações para execução das atividades.

Estas variáveis servem a responder as perguntas de pesquisa desta tese e serão trabalhadas nos resultados apresentados no próximo capítulo. No questionário ainda foram colocadas questões abertas, cujas respostas foram utilizadas apenas nas discussões para auxiliar a explicar os resultados.

5.1.7 Ameaças

Esta seção discute as ameaças de validade destes experimentos com sujeitos e as maneiras em que as abordamos. Três tipos de ameaças de validade foram identificadas: *1)* validade interna, referindo-se às inferências causa-efeito durante a análise; *2)* validade externa, quanto à generalização dos resultados para diferentes contextos; e *3)* Validade da Construção, que trata a concordância entre a concepção do experimento e o procedimento de medição.

5.1.7.1 Validade Interna

Sujeitos:

Existem várias ameaças de validade interna que se relacionam com os sujeitos participantes deste experimento

- **Disposição do participante** para o experimento, já que o desempenho do participante durante uma atividade é influenciada pela sua disposição no momento. Este pode estar disposto e animado ou cansado e chateado e isto pode interferir em sua concentração e capacidade de compreensão. Nos experimentos com grupos de alunos, os dois experimentos foram realizados no mesmo dia e comparados entre si, o que minimiza os impactos desta ameaça. Além do mais, foram definidos vários

dias e horários para participação dos alunos nos experimentos, para que estes escolhessem o melhor horário. Mas, os profissionais realizaram os dois experimentos em dias distintos e a disposição pode ter variado interferindo no seu desempenho em cada grupo. Para tentar minimizar esta ameaça, as atividades foram passadas com uma antecipação de 15 à 30 dias da data limite de entrega que foi negociada e o participante pode escolher o melhor horário e local para realizar a atividade. Um outro fator que ameniza esta ameaça é o fato de que todos os participantes eram voluntários e foram informados que poderiam desistir a qualquer momento.

- **Familiaridade do participante com as documentações** (Livro de Receitas e tradicional) e ambiente utilizado pode interferir nos resultados. Foi realizado um breve treinamento sobre as documentações e contato com o ambiente utilizado, onde o aprendizado e adaptação podem variar de participante para participante e interferir nos resultados.

- **Subjetividade** da definição sobre quais elementos de código são **uteis, complexos ou importantes** para a instanciação do *framework* na atividade, sob o ponto de vista do participante. Uma vez que os sujeitos participantes podem considerar diferentes elementos relevantes para a atividade, variando conforme o nível de conhecimento, a solução e compreensão apresentada sobre o *framework* e tecnologias necessárias para executar a(s) atividade(s). Para mitigar esta ameaça, os dados serão comparados entre os resultados do mesmo sujeito, variando apenas o uso do documento.

- Os **sujeitos** podem ter tido algum **conhecimento sobre o objetivo experimental**. Para amenizar o impacto desta ameaça, nenhum sujeito envolvido com a pesquisa participou dos experimentos. Os sujeitos não estavam familiarizados com as questões de pesquisa ou hipóteses (embora possam ter adivinhado) e não foram informados sobre qual documentação é resultado desta pesquisa.

Atividades:

- A **complexidade das tarefas** podem impactar no tempo e taxa de acerto. Serão definidas atividades com graus de dificuldade semelhantes para um mesmo experimento. O grau de dificuldade estará ligado a complexidade da atividade (conhecimento necessário), quantidade de linhas de código, classes e métodos e os tipos de interação com o *framework* que devem ser estabelecidas. Mas, a complexidade é algo que pode variar de desenvolvedor para desenvolvedor, não sendo totalmente controlada.

- As **atividades** podem ter sido **difíceis para o experimento**, então nenhum grupo teria bom desempenho em termos da taxa de acerto e tempo gasto, chegando ao

limite do tempo. O que pode prejudicar a medição do tempo e taxa de acerto. Para o experimento com profissionais, em que não foi dado um tempo limite para realizar as atividades e o nível de experiência dos sujeitos era avançado, o efeito da ameaça foi minimizado. Para os participantes alunos, que apresentam pouca experiência e tempo limite para realizar a atividade, a complexidade da atividade deve ser condizente ao nível de conhecimento destes e o tempo que terão para realizar a atividade. A fim de amenizar esta ameaça, todas as atividades foram testadas previamente com um desenvolvedor, que executou todas dentro do tempo limite.

Diversos:

- **O tamanho e conteúdo das duas documentações** pode influenciar na facilidade de consulta ou uso destas. As duas documentações não apresentam as informações sobre os *hot-spots* da mesma maneira e apresentam informações diferentes. Não existe uma definição sobre o que uma documentação de *framework* deve conter. Mas, ao criar os livros de receita, tentou-se criar uma documentação com volume próximo ao da documentação tradicional. Ou, limitar o tamanho da documentação tradicional considerando a parte com informações relevantes para as atividades. Assim, o esforço para consulta não seria tão influenciado pelo volume do documento.

- **Efeitos experimentador** (pesquisador) pode ocorrer, uma vez que o experimentador está envolvido com a proposta e tem interesse que esta seja bem sucedida, este pode influenciar nos experimentos ao instruir os sujeitos participantes, escolher as perguntas realizadas, selecionar os experimentos etc. Para minimizar este impacto, alguns procedimentos experimentais foram criados e seguidos. A pesquisadora deixou claro no documento de adesão dos sujeitos participantes sobre a importância da imparcialidade e que estes devem ser sinceros em suas respostas. A pesquisadora deve descartar as atividades do participante ao perceber algum erro no processo de coleta dos dados, avaliação e/ou execução da atividade. Tanto que, um experimento com 15 alunos, após 1h e 30min de experimento, foi descartado. Pois, se percebeu falhas nas configurações das maquinas onde seriam realizados os experimentos. Para manter a imparcialidade, os sujeitos participantes não estavam familiarizados com as questões de pesquisa.

5.1.7.2 Validade Externa

A generalização de nossos resultados pode ser dificultada pela representatividade limitada dos sujeitos e as tarefas.

- O ideal seria que a **amostra de sujeitos** representasse o todos os possíveis usuários da abordagem em um cenário real. A amostra é variada contando com alunos,

sujeitos da academia e da indústria de software. Porém, a maioria dos sujeitos são alunos que não apresentam experiência mais aprofundada com desenvolvimento. Conseguir a participação de profissionais é difícil (CORNELISSEN; ZAIDMAN; DEURSEN, 2011), mais de 50 profissionais da rede de contado dos pesquisadores foram contactados e apenas 15 participações foram efetivas, sendo que apenas 6 são da indústria. A amostra de sujeitos talvez não represente o perfil de possíveis usuários finais de uma abordagem como esta.

- As **atividades não cobrem todas as possibilidades** de instanciações de *framework*. Foram definidas 6 atividades, que por serem comparadas aos pares, cada par precisou apresentar complexidades próximas e portanto a mesmas necessidades de instanciação. As seis atividades não cobrem todas as possibilidades de instanciação de *framework*. Cobrir todas as possibilidades exigiria a definição de muitas atividades e para um experimento com execução de atividades reais, o tempo necessário para realizar as atividades tende a ser maior e portanto realizar muitas atividades em um experimento seria incompatível com a disponibilidade de tempo dos sujeitos participantes.

5.1.7.3 Validade da Construção

- O **primeiro contato** com o *framework* **pode influenciar no segundo contato**. É importante minimizar esta influência durante a segunda atividade do mesmo experimento. Por isso, definiu-se atividades que explorassem *hot-spots* e trechos de códigos diferentes entre as atividades. Porém, por se tratar do mesmo *framework* e até o mesmo sistema alvo de alterações, esta influencia não é possível de ser totalmente controlada. Um exemplo disto, é que para os experimentos com o *framework* JHotDraw, utilizou-se o sistema JModeller, que sofreu alterações para a primeira e segunda atividade. Logo, o desenvolvedor já teve contato com a distribuição dos pacotes, onde se encotrava a main() e outras informações sobre o código.

- **O código do sistema** JModeller foi **disponibilizado** para os dois grupos nos experimentos com profissionais e alunos da Instituição 2 e apresentavam exemplos de funcionalidades que poderiam ajudar o desenvolvedor durante as atividades definidas para o *framework* JHotDraw. Os desenvolvedores durante as atividades podem preferir consultar os códigos fontes das funcionalidade do que os documentos para realizar as atividades. E, como as atividades de desenvolvimento real envolvem consulta de código e alteração, não é possível limitá-lo. Mas, todos foram avisados que estas eram atividades para avaliar as documentações, que tentassem utilizá-las e observar o uso destas. As perguntas sobre as documentações irão ajudar a definir se estas foram úteis. Podendo ter influenciados nas taxas de acerto e tempo de execução.

A possibilidade de ocorrência destas ameaças durante os experimentos será discutida junto aos resultados no próximo capítulo.

5.1.8 Metodologia de Análise dos Dados

Para os três experimentos, utilizou-se o teste Mann-Whitney para análise do tempo de execução a fim de responder à pergunta P1. O teste de Friedman foi realizado para as demais variáveis a fim de responder as perguntas P2 à P5. A variável independente desta pesquisa é o uso das documentações pelos sujeitos. Para mitigar as ameaças ligadas aos níveis de conhecimento dos sujeitos e para não rejeitar nenhuma participação que seria necessária para o nivelamento dos grupos, resolveu-se realizar a comparação entre os resultados de cada sujeito, que hora atuaram como grupo Controle, hora como grupo Receitas. Logo todos os sujeitos participaram dos dois grupos em atividades distintas, em que se tentou ter a mesma complexidade. Logo estes grupos não precisam ser discriminados, e nem aplicada a randomização estratificada.

5.1.8.1 Teste de Friedman

O teste de Friedman é uma alternativa não-paramétrica à ANOVA unidirecional com medidas repetidas. Este irá considerar os resultados por escala, grupos e sujeitos no teste. É usado para testar diferenças entre grupos quando a variável dependente a ser medida é ordinal. Também pode ser usado para dados contínuos que violaram as suposições necessárias para executar a ANOVA unidirecional com medidas repetidas (por exemplo, dados que têm desvios acentuados da normalidade).

Friedman usa o p-value para determinar se alguma das diferenças entre as medianas são estatisticamente significativas. Para determinar se alguma das diferenças entre as medianas são estatisticamente significativas, compare o valor de p com o seu nível de significância para avaliar a hipótese nula. A hipótese nula afirma que as medianas da população são todas iguais. Normalmente, um nível de significância (denotado como alfa) de **0,05** é geralmente recomendado. Um nível de significância de 0,05 indica um risco de 5% de concluir que existe uma diferença quando não há diferença real.

5.1.8.2 Teste de Mann-Whitney

Para os testes de Mann-Whitney, considera-se duas populações P1 e P2 das quais não tem-se informações a respeito de suas distribuições, mas as variáveis envolvidas tenham uma escala de medida pelo menos ordinal. Ou seja, pode-se abordar o caso de variáveis aleatórias qualitativas ordinais ou quantitativas. Considera-se também duas amostras independentes das duas populações. O objetivo é testar se as distribuições são iguais em localização, isto é, se uma população tende a ter valores maiores do que a outra, ou se elas têm a mesma mediana. Este teste é chamado Mann-Whitney. Este teste pode

ser utilizado desde que se tenha uma distribuição similar entre as duas populações. Não utilizou-se o teste t de Student, por que os dados não apresentam distribuição normal.

Além dos testes de Friedman e Mann-Whitney, será realizada uma análise qualitativa dos percentuais relativos às perguntas de pesquisa e das respostas abertas realizadas pelos participantes para estas perguntas. No próximo capítulo serão apresentados os resultados

.

5.1.9 Aspectos Éticos

O projeto foi analisado pelo Comitê de Ética em Pesquisa com Seres Humanos. Todos os sujeitos participantes foram esclarecidos sobre a pesquisa e os que aceitarem confirmaram sua adesão por meio do Termo de Consentimento Livre e Esclarecido. Veja o modelo no Anexo 5. A equipe responsável pelo estudo se comprometeu a manter em sigilo absoluto a identidade dos sujeitos participantes incluídos no estudo, inclusive na publicação dos resultados. Todas as dúvidas referentes ao projeto de pesquisa foram esclarecidas. Os voluntários estiveram livres para retirar o seu consentimento de participação no projeto sem nenhum prejuízo e seriam informados caso isto ocorresse.

5.2 Resultados

Nesta seção serão apresentados os resultados das perguntas individuais. Isto permite uma análise mais precisa e aprofundada das forças e fraquezas de uma abordagem (WETTEL; LANZA; ROBBES, 2011). A análise será apresentada por experimento, para então realizarmos uma análise coletiva dos dados.

5.2.1 Experimento com Alunos - Instituição 1

A seguir serão apresentados os resultados para o experimento realizado com os 14 alunos da Instituição 1, utilizando o *framework* JFace e SWT para *plug-ins* Eclipse. Os resultados dos testes estatísticos aplicados foram consolidados na Tabela 10.

5.2.1.1 Tempo de execução

O tempo para execução da atividade foi coletado para cada sujeito, sendo que este foi delimitado por um tempo de 50 minutos por atividade. Como se pode ver no boxplot da Figura 26, a distribuição dos dois grupos não apresenta diferenças significativas na forma da distribuição, atendendo a premissa para aplicação do teste de Mann-Whitney. O resultado do teste apresentou um **p-value de 0,07881** que é **superior**, porém próximo, ao nível de significância de 0,05. Isto significa que a expectativa de erro em aceitar a diferença entre o grupo Receitas e o grupo Controle seria de 8%, o que é razoavelmente baixo. Entretanto considerando um padrão comumente adotado, não se pode afirmar que

Tabela 10 – Resultados dos experimento com alunos da Instituição 1.

Critério		
	W (Mann-Whitney)	p-value
Tempo de Execução	67	0.07881
	χ^2 (Friedman)	p-value
Tempo de Execução	6	**0.01431**
Taxa de Acerto	12	**0.000532**
Utilidade da Informação	4.5	**0.03389**
Complexidade da Atividade	5.4444	**0.01963**
Utilidade da Navegação	4.5	**0.03389**
Satisfação com Documentação	1.9231	0.1655
Facilidade de Localização	1	0.3173
Clareza da Informação	0.69231	0.4054

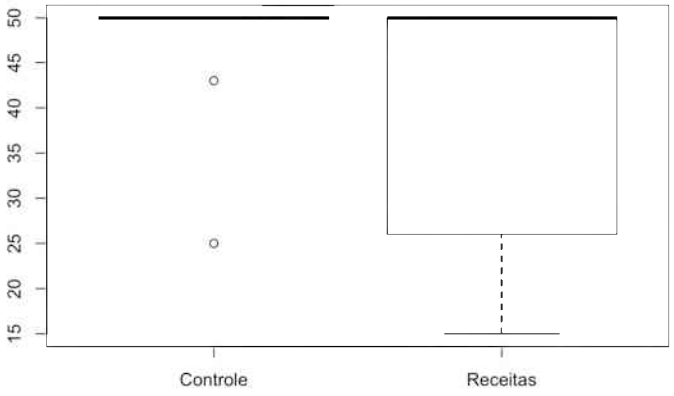


Figura 26 – Boxplot - Tempo de execução - Experimento com alunos da Instituição 1.

houve uma diferença significativa entre os grupos. Em uma análise qualitativa, o grupo Receitas apresentou uma quantidade de resultados relativos ao tempo menores, como se observa no boxplot.

5.2.1.2 Taxa de acerto

Após realizar as atividades, os códigos resultantes das atividades foram coletados para avaliação. A taxa de acerto foi medida avaliando os sistemas resultantes por meio de caso de uso executado e análise do código fonte, conforme um score de 0 à 4 que seguiu os seguintes critérios para os três experimentos:

- 0 : Não conseguiu realizar nada da atividade.
- 1 : Concluiu de 1% à 25% da atividade.

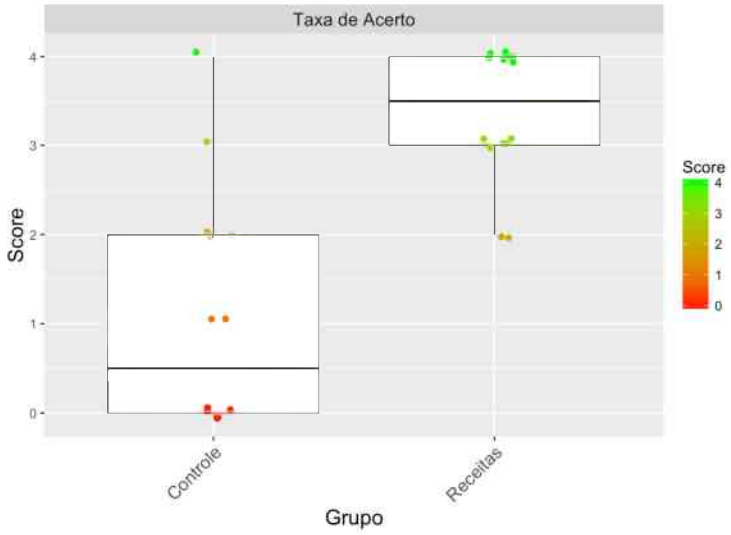


Figura 27 – Boxplot - Taxa de acerto do experimento com alunos da Instituição 1.

- 2 : Concluiu de 26% à 50% da atividade.
- 3 : Concluiu de 59% à 75% da atividade.
- 4 : Concluiu de 75% a 100% da atividade

O resultado para o teste de Friedman para as taxas de acerto atribuídas apresentou um **p-value de 0.000532** que **é menor** que o nível de significância e permite afirmar que o a taxa de acerto foi significantemente diferente para os dois grupos. Como se pode ver no boxplot da Figura 27, a mediana, primeiro e segundo quartil do grupo Receitas são superiores a mediana do grupo Controle, que **indica significantes maiores taxas de acerto para os sujeitos** do grupo Receitas que utiliza o livro de receitas.

5.2.1.3 Utilidade das informações

As demais perguntas de pesquisa foram respondidas por meio do questionário que os sujeitos responderam após realizar cada atividade. Estas perguntas foram medidas por meio da escala Likert:

- 2: Concordo Fortemente
- 1: Concordo
- 0: Neutro - Não soube opinar
- -1: Discordo
- -2: Discordo Fortemente

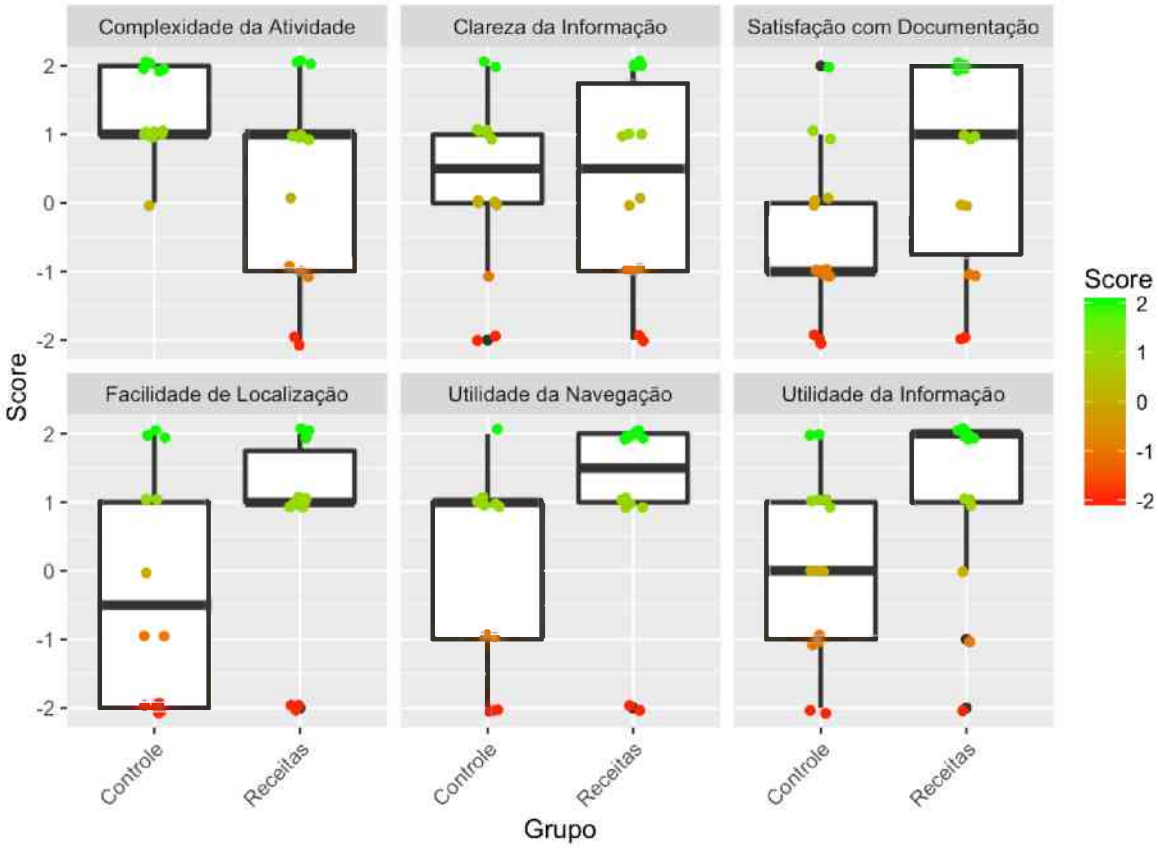


Figura 28 – Boxplot - Resultados do experimento com alunos da Instituição 1.

Os sujeitos foram questionados sobre a utilidade das informações presentes nos dois documentos (livro de receitas e tradicional) para realizar as atividades. Responderam conforme o boxplot da Figura 28. O resultado para o teste de Friedman apresentou um **p-value de 0.03389** que é **menor** que o nível de significância e permite afirmar que houve uma diferença significativa na percepção da utilidade das informações. Pelas mediana dos boxplots, observa-se que o **grupo Receitas apresentou uma percepção de utilidade maior.**

5.2.1.4 Complexidade da atividade

Os sujeitos foram questionados sobre a complexidade das atividades percebida por eles. Responderam a seguinte pergunta: Esta atividade foi COMPLEXA de se realizar? O resultado para o teste de Friedman obteve um **p-value de 0.01963** que é **menor** que o nível de significância e permite afirmar que o a percepção de complexidade da atividade foi significantemente diferente entre os grupos. Pela primeiro quartil e segundo quartil dos boxplots da Figura 28, se pode perceber que a **percepção de complexidade da atividade foi menor quando o sujeito estava utilizando o livro de receitas** no grupo Receitas.

5.2.1.5 Facilidade de localização da informações

Os sujeitos foram questionados sobre a facilidade para localizar as informações necessárias para realizar a atividade. Respoderam a seguinte pergunta: Foi FÁCIL LOCALIZAR as informações necessárias para instanciação no Documento? O resultado para o teste de Friedman obteve um **p-value de 0.3173** que é **superior** ao nível de significância. Portanto, pode-se afirmar que a percepção de facilidade na localização das informações necessárias não foi significantemente diferente entre os grupos, conforme também pode ser observado no boxplot da Figura 28.

5.2.1.6 Utilidade da navegação dos documentos

Os sujeitos foram questionados sobre a utilidade do mecanismo de navegação nos documentos. Responderam a seguinte pergunta: O mecanismo de navegação pelo documento foi útil para a atividade? O resultado para o teste de Friedman apresentou um **p-value de 0,03389**, **menor** que o nível de significância. Portanto, pode-se afirmar que o a percepção de utilidade do mecanismo de navegação teve uma diferença significante entre os grupos. No boxplot da Figura 28, se pode notar a mediana superior para o grupo Receitas, ou seja, **os sujeitos deste grupo apresentaram maior concordância sobre a utilidade do mecanismo de navegação**.

5.2.1.7 Clareza das informações

Os sujeitos foram questionados sobre a clareza das informações presentes nos documentos. Responderam a seguinte pergunta: as informações dispostas no documento estavam CLARAS, ou seja, facilmente compreensíveis e bem apresentadas? O resultado para o teste de Friedman apresentou um **p-value de 0.4054** que é **superior** ao nível de significância. Portanto, pode-se afirmar que o a percepção sobre a clareza das informações presentes no Livro de receitas não foi significamente diferente para os dois grupos. Veja o boxplot da Figura 28.

5.2.1.8 Satisfação com o uso do documentos

Os sujeitos foram questionados sobre a satisfação com o uso dos documentos. Respoderam a seguinte pergunta: Você ficou SATISFEITO com o uso do Documento para realizar esta atividade? Satisfação: Sensação agradável que sentimos quando as coisas correm à nossa vontade ou se cumprem a nosso contento. O resultado para o teste de Friedman apresentou um **p-value de 0.1655** que é **superior** ao nível de significância. Portanto, pode-se afirmar que o a percepção de satisfação com o uso dos documentos não foi significamente diferente para os dois grupos.Veja o boxplot da Figura 28.

5.2.2 Experimento com Alunos – Instituição 2

A seguir serão apresentados os resultados para o experimento realizado com os alunos da Instituição 2, utilizando o *framework* JHotDraw. Os resultados dos testes estatísticos aplicados foram consolidados na Tabela 11.

Tabela 11 – Resultados dos experimento com alunos da Instituição 2.

Critério		
	W (Mann-Whitney)	p-value
Tempo de Execução	120	0.3506
	χ^2 (Friedman)	p-value
Tempo de Execução	1	0.3173
Taxa de Acerto	2	0.1573
Utilidade da Informação	5.3333	**0.02092**
Complexidade da Atividade	0.66667	0.4142
Utilidade da Navegação	3.6	**0.05778**
Satisfação com Documentação	6.4	**0.01141**
Facilidade de Localização	5.4444	**0.01963**
Clareza da Informação	4.4545	**0.03481**

5.2.2.1 Tempo de execução

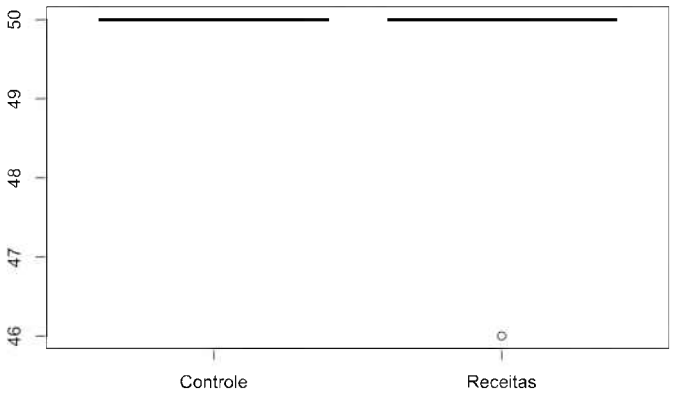


Figura 29 – Boxplot - Tempo de Execução do experimento com alunos da Instituição 2.

O tempo para execução da atividade foi coletado para cada sujeito seguindo o mesmo processo do experimento com os alunos da Instituição 1. O resultado para o teste de Mann-Whitney para o tempo de execução apresentou um **p-value de 0.3506** que é **superior** ao nível de significância. Logo, pode-se afirmar que não houve diferença significativa no tempo de execução das atividades entre os grupos, como se pode notar pelo boxplot da Figura 29.

5.2.2.2 Taxa de acerto

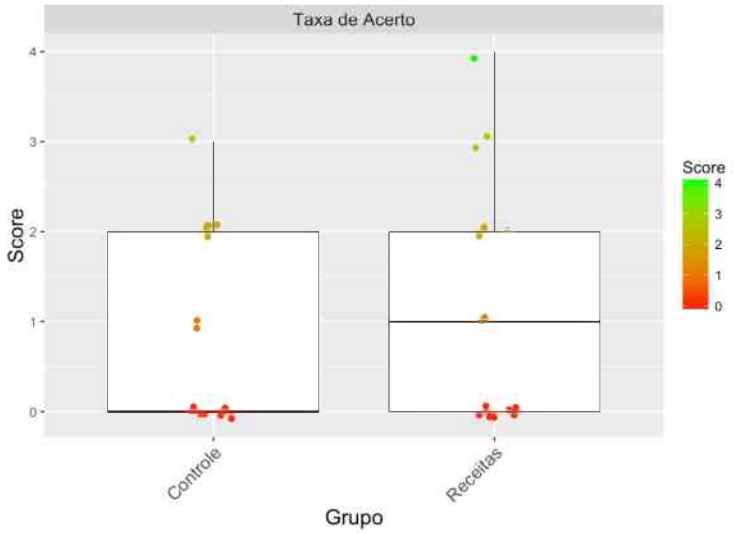


Figura 30 – Boxplot - Taxa de acerto do experimento com alunos da Instituição 2.

A taxa de acerto foi medida avaliando seguindo o mesmo processo do experimento com alunos da Instituição 1. O resultado para o teste de Friedman para as taxas de acerto atribuídas apresentou um **p-value de 0.1573** que é **superior** ao nível de significância, portanto não houve diferença significante entre os resultados dos dois grupos, como se observa no boxplot da Figura 30.

5.2.2.3 Utilidade das informações

As perguntas de pesquisa a seguir foram respondidas por meio do questionário utilizando uma escala Likert apresentada nos resultados do experimento com alunos da Instituição 1. O mesmo processo e questionário foi aplicado para os três experimentos.

Sobre a utilidade das informações dos dois documentos (Livro de Receitas e Tradicional), o resultado para o teste de Friedman apresentou um **p-value de 0.02092** que é **menor** que o nível de significância e permite afirmar que houve uma diferença significante entre os resultados dos dois grupos. Conforme o boxplot da Figura 31, a mediana, primeiro e segundo quartil do boxplot do grupo Receitas são superiores aos do grupo Controle. Logo, o **a percepção de utilidade das informações presentes no livro de receitas foi significantemente maior**.

5.2.2.4 Complexidade da atividade

Sobre o grau de complexidade das atividades percebido, o resultado para o teste de Friedman obteve um **p-value de 0.4142** que é superior ao nível de significância, permitindo afirmar que não houve diferença significante entre a percepção de complexidade entre os grupos, como se pode ver no boxplot da Figura 31.

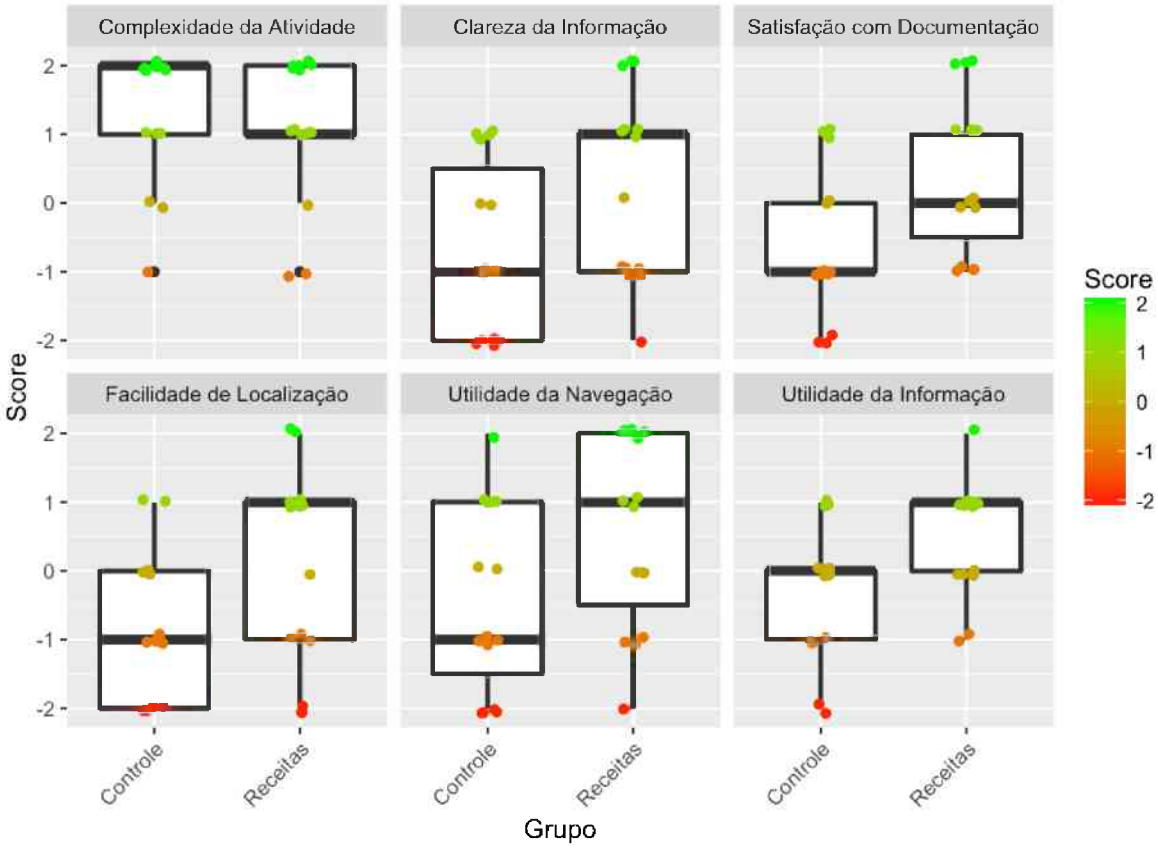


Figura 31 – Boxplot - Resultados do experimento com alunos da Instituição 2.

5.2.2.5 Facilidade Localização da Informações

Sobre a facilidade para localizar as informações necessárias para realizar a atividade, o resultado para o teste de Friedman obteve um**p-value de 0.01963** que é **inferior** ao nível de significância. Portanto, pode-se afirmar que houve uma diferença significativa entre os resultados dos dois grupos. **A facilidade de localização foi significantemente maior para os sujeitos no grupo Receitas**. Como se pode notar no boxplot da Figura 31, a mediana é superior para este grupo.

5.2.2.6 Utilidade da Navegação dos Documentos

Sobre a utilidade do mecanismo de navegação nos documentos, o resultado para o teste de Friedman apresentou um **p-value de 0.05778** que considerou-se **igual** ao nível de significância de 0,05. Portanto, pode-se afirmar que houve uma diferença significante na percepção de utilidade do mecanismo de navegação entre os grupos. Pelo boxplot da Figura 31, se pode notar que a mediana e o primeiro quartil do grupo Receitas superiores aos do grupo Controle, **indicando que a utilidade percebida para o mecanismo de navegação foi maior para o grupo Receitas**.

5.2.2.7 Clareza das informações

Sobre a clareza das informações presentes nos documentos, o resultado para o teste de Friedman apresentou um **p-value de 0.03481** que é **inferior** ao nível de significância. Portanto, pode-se afirmar que houve uma diferença significante entre os resultados dos dois grupos. A mediana do grupo Receitas é superior à do grupo Controle, como se pode ver no boxplot da Figura 31. Logo, **a percepção sobre a clareza das informações presentes no Livro de Receitas foi significantemente maior** do que a percepção de clareza das informações presentes no documento Tradicional.

5.2.2.8 Satisfação com o uso do documentos

Sobre a satisfação com o uso dos documentos, o resultado para o teste de Friedman apresentou um **p-value de 0.01141** que é **inferior** ao nível de significância. Portanto, pode-se afirmar que houve uma diferença significante entre os resultados dos dois grupos. **Onde a percepção sobre a satisfação com o uso do Livro de Receitas no grupo Receitas foi significantemente maior** do que a percepção de satisfação com o uso do documento Tradicional. Como se pode notar pela mediana, primeiro e segundo quartil do grupo Receitas superiores aos do grupo Controle, no boxplot da Figura 31.

5.2.3 Experimento com Profissionais

A seguir serão apresentados os resultados para o experimento realizado com os profissionais graduados utilizando o *framework* JHotDraw. Os resultados dos testes estatísticos aplicados foram consolidados na Tabela 12.

Tabela 12 – Resultados dos experimento com Profissionais.

Critério		
	W (Mann-Whitney)	p-value
Tempo de Execução	94.5	0.4679
	χ^2 (Friedman)	p-value
Tempo de Execução	0.6	0.4386
Taxa de Acerto	0.33333	0.5637
Utilidade da Informação	0.14286	0.7055
Complexidade da Atividade	0.11111	0.7389
Utilidade da Navegação	4.5714	**0.03251**
Satisfação com Documentação	4.5	**0.03389**
Facilidade de Localização	4.4545	**0.03481**
Clareza da Informação	1	0.3173

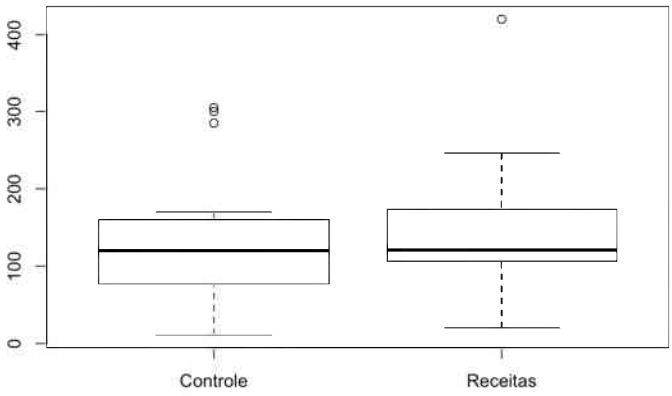


Figura 32 – Boxplot - Tempo de execução do experimento com Profissionais.

5.2.3.1 Tempo de execução

O tempo para execução da atividade foi coletado para cada sujeito seguindo o mesmo processo do experimento com os Profissionais. O resultado para o teste de Mann-Whitney para o tempo de execução apresentou um **p-value de 0.4679** que é **superior** ao nível de significância. Logo, pode-se afirmar que não houve diferença significante no tempo de execução das atividades entre os grupos, como se pode notar pelo boxplot da Figura 32.

5.2.3.2 Taxa de acerto

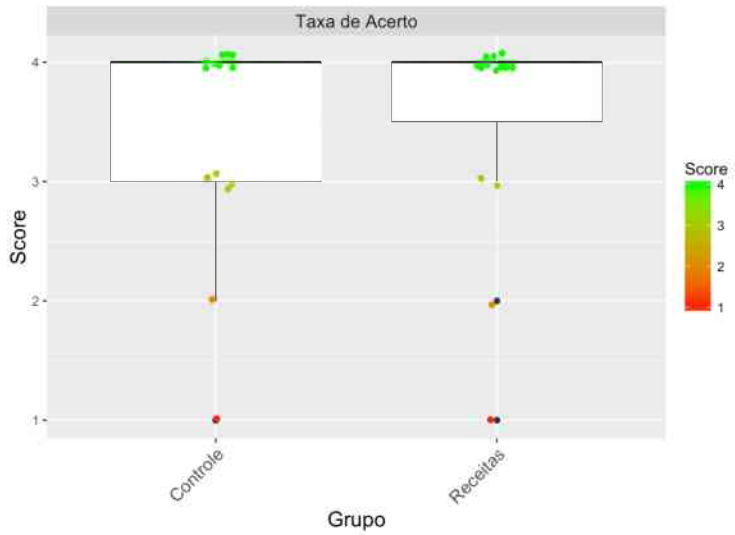


Figura 33 – Boxplot - Taxa de acerto do experimento com alunos Profissionais.

A taxa de acerto foi medida avaliando seguindo o mesmo processo do experimento com profissionais. O resultado para o teste de Friedman para as taxas de acerto atribuídas apresentou um **p-value de 0.5637** que é **superior** ao nível de significância, portanto

não houve diferença significante entre os resultados dos dois grupos, como se observa no boxplot da Figura 33.

5.2.3.3 Utilidade das informações

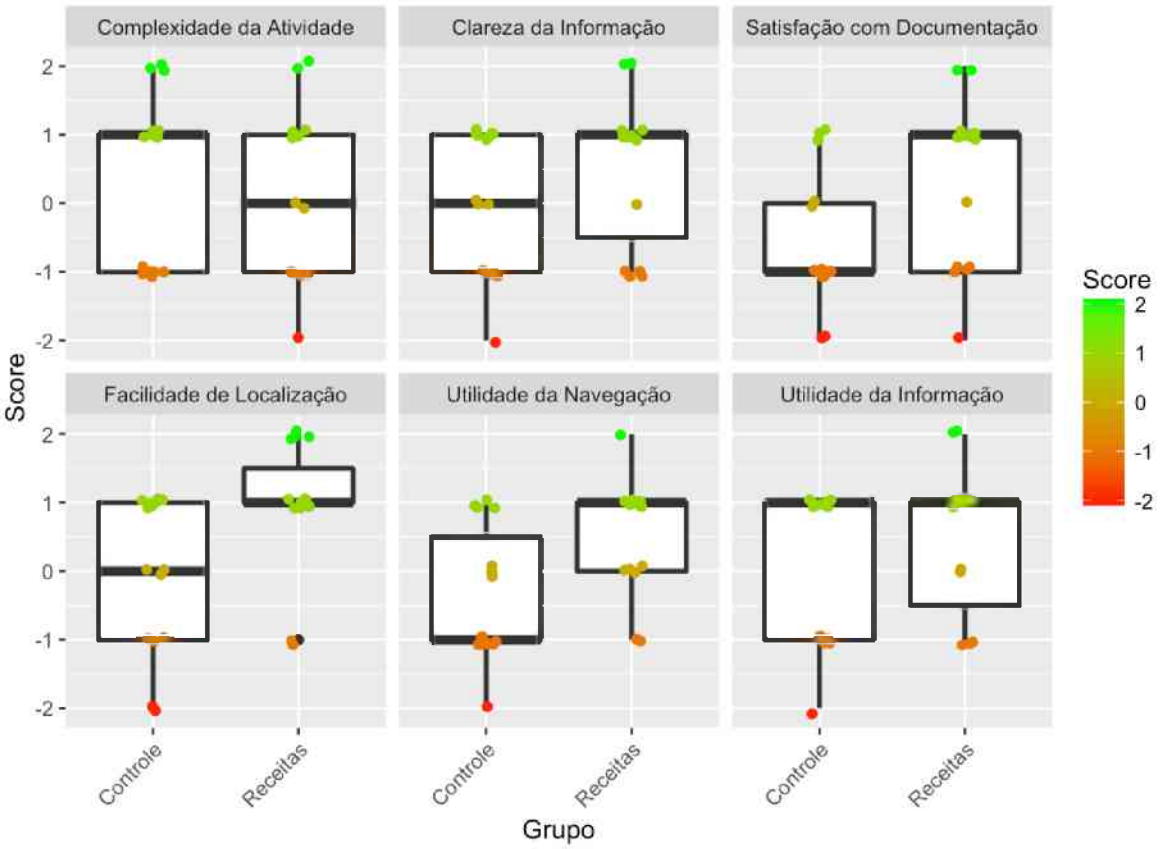


Figura 34 – Boxplot - Resultados do experimento com alunos Profissionais.

Sobre a utilidade das informações dos dois documentos (Livro de Receitas e Tradicional), o resultado para o teste de Friedman apresentou um **p-value de 0.7055** que é **superior** ao nível de significância, portanto não houve diferença significante entre os resultados dos dois grupos, como se pode ver no boxplot da Figura 34.

5.2.3.4 Complexidade da atividade

Sobre o grau de complexidade das atividades percebido, o resultado para o teste de Friedman obteve um **p-value de 0.7389** que é **superior** ao nível de significância, permitindo afirmar que não houve diferença significante entre a percepção de complexidade entre os grupos, como se observa no boxplot da Figura 34.

5.2.3.5 Facilidade Localização da Informações

Sobre a facilidade para localizar as informações necessárias para realizar a atividade, o resultado para o teste de Friedman obteve um**p-value de 0.03481** que é **inferior** ao nível

de significância. Portanto, pode-se afirmar que houve uma diferença significativa entre os resultados dos dois grupos. **A facilidade de localização foi significantemente maior para os sujeitos no grupo Receita**. Como se observa no boxplot da Figura 34, a mediana, primeiro e segundo quartil são superiores aos do grupo Receitas.

5.2.3.6 Utilidade da Navegação dos Documentos

Sobre a utilidade do mecanismo de navegação nos documentos, o resultado para o teste de Friedman apresentou um **p-value de 0. 0.03251** que é **inferior** ao nível de significância. Portanto, pode-se afirmar que houve uma diferença significante na percepção de utilidade do mecanismo de navegação entre os grupos. A mediana, primeiro e segundo quartil do grupo Receitas são superiores aos do grupo Controle, veja o boxplot da Figura 34. **Este resultado indica que a utilidade percebida para o mecanismo de navegação foi maior para o grupo Receitas**.

5.2.3.7 Clareza das informações

Sobre a clareza das informações presentes nos documentos, o resultado para o teste de Friedman apresentou um p-value de 0.3173 que é superior ao nível de significância. Portanto não houve uma diferença significante entre os resultados dos dois grupos quando se trata da clareza das informações, veja o boxplot da Figura 34.

5.2.3.8 Satisfação com o uso do documentos

Sobre a satisfação com o uso dos documentos, o resultado para o teste de Friedman apresentou um **p-value de 0.03389** que é **inferior** ao nível de significância. Portanto, pode-se afirmar que houve uma diferença significante entre os resultados dos dois grupos. **Onde a percepção sobre a satisfação com o uso do Livro de Receitas no grupo Receitas foi significantemente maior** do que a percepção de satisfação com o uso do documento Tradicional, como se pode notar pela mediana no boxplot da Figura 34.

5.2.4 Conclusão dos Resultados

Nesta seção foram apresentados os resultados de pesquisa. Estes mostraram que o uso do livro de receitas para todas as variáveis apresentou resultados melhores ou sem significante diferença quando comparado ao documento tradicional do *framework*. Para tentar entender melhor os fatores que levaram a estes resultados, estes serão discutidos considerando as características de cada experimento, como perfil dos sujeitos, objeto de pesquisa e atividade aplicada. As Tabelas 13, 14 e 15 apresentam os percentuais relativos aos resultados de cada experimento, utilizados nas discussões a seguir.

Tabela 13 – Resultados percentuais do experimento com alunos da Instituição 1.

Grupo	Escala Likert	Utilidade Navegação	Utilidade Informação	Facilidade Localização	Clareza Informação	Satisfação Documentação	Complexidade Atividade
Receitas	-2	14,29%	7,14%	21,43%	14,29%	14,29%	14,29%
Receitas	-1	0,00%	7,14%	0,00%	21,43%	14,29%	21,43%
Receitas	0	0,00%	7,14%	0,00%	14,29%	14,29%	7,14%
Receitas	1	35,71%	21,43%	50,00%	21,43%	21,43%	35,71%
Receitas	2	50,00%	57,14%	28,57%	28,57%	35,71%	21,43%
Controle	-2	21,43%	14,29%	35,71%	14,29%	21,43%	0,00%
Controle	-1	21,43%	21,43%	14,29%	7,14%	35,71%	0,00%
Controle	0	0,00%	21,73%	7,14%	28,57%	21,43%	7,14%
Controle	1	50,00%	28,57%	21,43%	35,71%	14,29%	57,14%
Controle	2	7,14%	14,29%	21,43%	14,29%	7,14%	35,71%

Tabela 14 – Resultados percentuais do experimento com alunos da Instituição 2.

Grupo	Escala Likert	Utilidade Navegação	Utilidade Informação	Facilidade Localização	Clareza Informação	Satisfação Documentação	Complexidade Atividade
Receitas	-2	6,67%	0,00%	20,00%	6,67%	0,00%	0,00%
Receitas	-1	20,00%	13,33%	20,00%	33,33%	26,67%	13,33%
Receitas	0	20,00%	26,27%	6,67%	6,67%	26,67%	6,67%
Receitas	1	20,00%	53,33%	40,00%	33,33%	26,67%	33,33%
Receitas	2	33,33%	6,67%	13,33%	20,00%	20,00%	46,67%
Controle	-2	26,67%	13,33%	33,33%	33,33%	20,00%	0,00%
Controle	-1	26,67%	20,00%	33,33%	26,67%	46,67%	6,67%
Controle	0	13,33%	46,67%	20,00%	13,33%	13,33%	13,33%
Controle	1	26,67%	20,00%	13,33%	26,67%	20,00%	20,00%
Controle	2	6,67%	0,00%	0,00%	0,00%	0,00%	60,00%

5.3 Discussão

Nesta seção, os resultados serão discutidos por experimento para então discutir os resultados por variável de pesquisa. Para uma análise qualitativa dos resultados, serão consideradas as opiniões dos sujeitos. No questionário, para cada pergunta de pesquisa fechada (escala Likert) havia uma pergunta aberta optativa. Estas opiniões são relevantes para poder entender melhor os resultados apresentados.

5.3.1 Experimento Alunos - Instituição 1

Diferentemente dos demais experimentos, neste não foi disponibilizado o código fonte de um sistema que os usuários pudessem se basear para executar a atividade. Portanto,

Tabela 15 – Resultados percentuais do experimento com profissionais.

Grupo	Escala Likert	Utilidade Navegação	Utilidade Informação	Facilidade Localização	Clareza Informação	Satisfação Documentação	Complexidade Atividade
Receitas	-2	0,00%	0,00%	0,00%	0,00%	6,67%	6,67%
Receitas	-1	13,33%	26,67%	13,33%	26,67%	26,67%	40,00%
Receitas	0	26,67%	13,33%	0,00%	6,67%	6,67%	13,33%
Receitas	1	53,33%	46,67%	60,00%	53,33%	46,67%	26,67%
Receitas	2	6,67%	13,33%	26,67%	13,33%	13,33%	13,33%
Controle	-2	6,67%	6,67%	13,33%	6,67%	13,33%	0,00%
Controle	-1	46,67%	33,33%	20,00%	26,67%	53,33%	46,67%
Controle	0	20,00%	60,00%	20,00%	20,00%	13,33%	0,00%
Controle	1	26,67%	0,00%	46,67%	46,67%	20,00%	33,33%
Controle	2	0,00%	0,00%	0,00%	0,00%	0,00%	20,00%

o único apoio aos sujeitos foram as documentações. Porém, ambas as documentações apresentavam exemplos de código que poderiam ajudar os desenvolvedores nas duas atividades. Para este experimento a taxa de acerto foi significantemente melhor. O tempo de execução quando comparado apresentou p-value de 0,07 (Tabela 10), próximo de ser significante, apontando para um tempo melhor para o grupo Receitas. A complexidade da atividade percebida pelos sujeitos do grupo Receitas foi significantemente menor para os sujeitos no grupo Receitas, o que pode ser reflexo do uso da documentação que pode ter facilitado a execução das atividades.

A navegação foi considerada significantemente mais útil pelos usuários do livro de receitas, podendo ser explicado pelo fato de as receitas apresentarem todas as informações dos *hot-spots* e exemplos de uso destes para a característica, em um único tópico que apresenta ganchos para informações mais detalhadas. Por outro lado, a documentação tradicional do JFace apresenta tópicos para as características tratadas pelas duas atividades, mas o desenvolvedor deveria ler a documentação e continuar navegando nesta para encontrar todas as informações e os exemplos de código. Para a atividade 1.2, **grupo Controle**, o sujeito **A** respondeu: "*Com a documentação disponível não consegui encontrar as informações que precisava para realizar a atividade*". E, o sujeito **B** respondeu: "*Para um profisisonal creio que esteja muito bem apresentada, porém no meu caso não soube usar os métodos*". Para a atividade 1.1, no **grupo Receitas**, o sujeito **B** respondeu que o Livro de Receitas "*foi útil para encontrar os métodos que deveria usar*".

A documentação tradicional do JFace é uma documentação completa e detalhada, com exemplos de código e recursos de consulta. Isto pode explicar, porque os itens satisfação, facilidade de localização, e clareza da informação não apresentaram diferença significativa entre as documentações. Considerando que os livros de receitas são gerados semi-automaticamente, um resultado próximo aos da documentação tradicional pode ser

considerado satisfatório, pois estas são oficiais e normalmente envolvem o conhecimento de especialista para serem geradas. Observe o parecer do sujeito **C** na Atividade 1.2, **grupo Controle**, quando questionado sobre a satisfação ao utilizar a documentação tradicional respondeu: "*Essa documentação está bem formatada, completa e segue o padrão de documentação da Oracle, tornando as eventuais buscas mais fáceis devido à familiaridade com o formato da documentação*". O sujeito **C** foi o participante que obteve o melhor desempenho nos dois grupos. Estes pareceres reforçam que a documentação tradicional do JFace é completa e bem estruturada. Entretanto, para alguns sujeitos não foi fácil realizar uma consulta nesta documentação e localizar as informações e exemplos.

As opiniões apresentadas nas questões abertas confirmam as hipóteses iniciais de que os exemplos de código são úteis, auxiliam na instanciação de *frameworks*. Seguem algumas opiniões dos sujeitos sobre o Livro de Receitas (Receitas) e a documentação Tradicional (Controle). Para a Atividade 1.1, grupo **Receitas**, o sujeito **C** quando questionado sobre a utilidade das informações do livro de receitas, respondeu: "*Foi graças ao exemplo referente ao hot-spot que consegui finalizar a atividade*". Quando questionados sobre a utilidade dos exemplos, o sujeito **C** repetiu o parecer. Sobre a utilidade das informações, o sujeito D respondeu: "*Foi útil pelo fato de que, com os exemplos, foi mais fácil visualizar o que era preciso no código para o programa funcionar, facilitando a visualização das classes e do uso dos hot-spots.*"; E, o sujeito **A** respondeu: "*O exemplo de uso facilita o entendimento de como as classes se relacionam, de maneira rápida e direta sem a necessidade de navegar pela documentação por muito tempo para entender os relacionamentos*".

Enfim, o livro de receitas pode ter auxiliado na execução das atividades, levando a uma maior taxa de acerto, maior percepção de utilidade do documento e consequente menor percepção da complexidade da atividade, dada a sua objetividade em ligar um exemplo de código com uma característica de interesse.

5.3.2 Experimento Alunos - Instituição 2

As duas atividades realizadas no experimentos com alunos da Instituição 2 foram avaliadas como atividades complexas pelos sujeitos, considere o tempo e o nível de conhecimento dos alunos iniciantes apresentados na Seção 5.1.4.2. O p-value (Tabela 11) sobre a percepção de complexidade da atividade não demonstrou diferença significante entre os dois grupos. Em uma análise qualitativa, 80% dos sujeitos no **grupo Receitas** que utilizaram o Livro de Receitas perceberam a atividade complexa, 46,67% concordou fortemente e 33,33% concordou. No **grupo Controle**, este valor se repete, 80% perceberam a atividade complexa, 60% concordou fortemente e 20% concordou. Veja a Tabela 14. Os sujeitos que responderam as perguntas abertas do questionário demonstraram dificuldade em realizar a atividade. Observe algumas das respostas sobre a complexidade dos sujeitos em **ambos os grupos**: "*Tive dificuldade para realizar a tarefa no tempo estipulado*"; "*Atividade foi complexa, pois tive dificuldade em identificar como fazer a chamada para*

colorir o objeto"; "*Complicado de encontrar as informações para auxiliar no desenvolvimento da atividade*"; e, "*Não tenho grande domínio sobre Java, logo, tive dificuldades em realizar a atividade. Meu grande problema nesta atividade foi como "instanciar"o objeto a ser movimentado.*"

As atividades 2.1 e 2.2 realizadas nos experimentos com alunos da Instituição 2 são as atividades que envolvem o menor número de *hot-spots* e com o menor número de ações em relação às outras quatro atividades dos demais experimentos. Tanto o código fonte do sistema JModeller como o livro de receitas apresentam exemplos de características que usavam os *hot-spots* que deveriam ser usados nas duas atividades. Os sujeitos deveriam descobrir quais receitas tratavam das características que poderiam ajudar e entender como os *hot-spots* de interesse foram usados. Pelos questionários, observou-se que 86% dos sujeitos no grupo Receitas acessaram as receitas que poderiam os ajudar nestas atividades. Pelos códigos, foi possível perceber que os alunos com taxa de acerto acima de 3, apenas 26,67% dos sujeitos de ambos os grupos (ver Tabela 14) conseguiram descobrir os *hot-spots*, mas apenas 1 (6.67%) sujeito do grupo Receita conseguiu implementar o uso deles, que era uma invocação simples. Pelos exemplos, eles não foram capazes de descobrir como utilizar os métodos. Uma possível razão é que os exemplos eram mais complexos do que as atividades solicitadas, as quais eram simples atividades de instanciação de *frameworks*. Enfim, apesar de as atividades terem uma complexidade mínima necessária para caracterizar o uso de *frameworks*, a instanciação de *framework* é inerentemente complexa. Dado ao nível de experiência dos sujeitos, mesmo as atividades sendo minimamente complexas, ainda assim estas foram avaliadas como complexas por eles. Ou seja, um fator de sucesso na instanciação de *frameworks* tão ou mais importante que a qualidade da documentação, é a maturidade do desenvolvedor.

Sobre a taxa de acerto, 46,66% dos sujeitos no grupo Receita concluiram parcialmente a atividade e apenas 6,67% (um) sujeito **A** concluiu a atividade com sucesso. Apenas este sujeito terminou a atividade antes do tempo limite. Este único sujeito é um aluno que fez um curso de programação Java à parte dos cursos da instituição e possui o maior nível de conhecimento sobre o paradigma de orientação à objetos e uso de *frameworks* no grupo. No grupo Controle, 46,67% concluiu parcialmente a atividade e nenhum sujeito concluiu a atividade com sucesso no tempo limite definido, como se pode ver na Tabela 14.

A percepção de complexidade das atividades para os sujeitos neste experimento pode ter sido um fator impactante no tempo gasto e nas taxas de acerto. Os resultados dos dois grupos conforme o p-value da Tabela 11 foi considerado sem diferença significante. Mas, o fator pode ser as atividades e não o uso dos documentos.

O único sujeito **A** que concluiu uma das atividades e apresentou maior conhecimento em desenvolvimento JAVA, respondeu o seguinte sobre o mecanismo de navegação do Livro de **Receitas**: "*Pelo fato de estar dividido por atividades possíveis, foi só buscar uma atividade que tivesse o mesmo conceito da atividade proposta*"; Sobre a utilidade das

informações do Livro de **Receitas**, ele respondeu: "*Sem as informações que estavam lá não daria para fazer, pois não conhecia o framework*". Este sujeito não conseguiu concluir a atividade no grupo Controle e quando questionado sobre a clareza da **documentação Tradicional**, respondeu: "*a documentação tem quase nenhuma informação*". Quando questionado sobre a satisfação ao utilizar a documentação tradicional, respondeu: ' '*não achei nada*".

Um outro sujeito **B**, no grupo **Controle**, atividade 2.1, respondeu sobre a satisfação de utilizar a documentação tradicional: "*A receita da aplicação anterior era mais fácil*". O sujeito **C** do grupo **Controle**, atividade 2.2, respondeu sobre a satisfação: "*Não consegui compreender está documentação*". O sujeito **D**, deste mesmo grupo **controle** respondeu: "*Faltou exemplos mais claros de código*". Este mesmo sujeito **D** respondeu a mesma pergunta sobre o livro de receitas: "*Pois possuo pouca experiência na linguagem, porem a documentação está bem estruturada*". O sujeito **E** no grupo **controle** respondeu: "*A primeira estava mais fácil de localizar as classes/exemplos do que esta*". Um sujeito **F**: respondeu sobre a utilidade do mecanismo de navegação e sobre a Satisfação da documentação Tradicional: "*A interface do Javadoc é mais intuitiva para mim*". Não houve outros pareceres favoráveis à documentação tradicional. Um sujeito **G**, no grupo **Receitas** respondeu sobre a utilidade das informações: "*Dificuldade na interpretação da utilização da receita*". Estas respostas ajudam a justificar o porquê do nível de satisfação, facilidade de localização, utilidade da informação, utilidade da navegação e clareza da informação tenderem significantes melhores resultados no grupo Receitas.

Sobre os exemplos de código, os sujeitos do grupo **Receitas** deram os seguintes pareceres: "*Através deles que identifiquei onde fazer alteração*"; "**Facilitou o caminho para chegar a solução**"; "*utilizei-as como exemplo para tentar solucionar meu problema*"; "*Facilita o entendimento*"; e, "*Os exemplos de uso estavam claros e de fácil entendimento*". Estes pareceres reforçam a utilidade dos exemplos para o aprendizado de uso dos *frameworks*.

5.3.3 Experimento com Profissionais

Para os experimentos com profissionais, o tempo foi ilimitado, e não apresentou diferença significante entre os dois grupos. Os participantes tomaram o tempo necessário para finalizarem as atividades e no geral apresentaram boas taxas de acerto em ambos os grupos. Não houve diferença significante na taxa de acerto, quando se compara o grupo Receitas com o grupo Controle. Este resultado pode ter sido gerado pelo fato de que a maioria dos sujeitos deste experimento, para ambos os grupos, se apoiaram nos exemplos de código presentes no sistema a ser alterado ou nos exemplos disponibilizados dentro do código do *framework*, dando a mesma base para ambos os grupos. Este fato será discutido a seguir.

Segundo os resultados, os sujeitos ao utilizarem o livro de receitas tiveram uma maior percepção de facilidade de localização das informações, maior utilidade do mecanismo de navegação e maior satisfação ao utilizar o livro de receitas comparado ao documento tradicional. Seguem alguns pareceres que podem ajudar entender estes resultados. Quando questionados sobre a navegabilidade, no grupo **Receita** os sujeitos responderam: *"O documento ficou com aspecto mais limpo, trazendo apenas aquilo que realmente era necessário/selecionado"*. No grupo **Controle** sobre a documentação tradicional, os sujeitos responderam: *"O mecanismo de navegação facilita encontrar a documentação das classes necessárias para desenvolver a atividade"*. Entretanto uma crítica foi: *"Não serviu de nenhuma utilidade pois para realizar a modificação proposta a documentação não forneceu dados necessários"*. Sobre facilidade de localização, o grupo **Receitas** não apresentou respostas. O grupo **Controle** apresentou uma resposta: *"Somente olhando para o documento eu acho que eu demoraria muito mais para executar a atividade. O que adiantou e muito a execução foi tomar uma outra classe já implementada como exemplo"*.

Sobre satisfação, no grupo Receitas, os sujeitos apresentaram os seguintes pareceres: *"Sentime mais confortável analisando o código e elaborando por analogia e tentativa/erro, mas isso pode ser uma característica particular minha. A utilidade do documento que mais me foi útil para me mostrar as diversas opções de se instanciar um novo objeto Tool para ser incluído no paleta."*. Seguem algumas das respostas dos sujeitos no grupo **Controle**: *"Eu não fiquei satisfeita com o uso do documento visto que o que me ajudou de fato a realizar a atividade foram exemplos de código encontrados no código fonte do framework. O documento só me ajudou a descobrir a existência de tais exemplos"*; *"O documento quase não ajudou. O que consegui fazer foi praticamente só lendo o código fonte"*. Foram vários relatos com a mesma opinião, que levaram à conclusão de que vários sujeitos utilizaram ambos os documentos apenas para localizar os exemplos de código no sistema que poderiam ajudá-los, e utilizaram estes exemplos para entender melhor a solução.

Neste experimento, foi utilizado o *framework* JHotDraw que apresenta exemplos de código dentro do *framework*, e estes exemplos foram mantidos como tipo de informação útil. Até mesmo para saber se a documentação substitui a consulta ao código. Já que falou-se tanto da utilidade dos sistemas como exemplos de código para o aprendizado de *frameworks*. Diante dos pareceres feitos pelos sujeitos, tem-se a percepção de que os exemplos de código presentes no sistema alvo das alterações (JModeller) e nos exemplos do *framework* foram úteis para os dois grupos. E, que ambos os documentos guiaram os sujeitos à acharem os exemplos de código. Segue uma das respostas do grupo **Receita**: *"A documentação serviu, particularmente, para me mostrar como instanciar um objeto Tool conforme minha necessidade"*. No grupo **Controle**, os sujeitos relataram: *"A documentação só foi útil para a descoberta da existência de exemplos de código de uso do* framework *dentro do próprio framework"*; *"A parte da documentação que me ajudou foram os samples. A partir deles encontrei por onde tudo começa e entendi a lógica da*

aplicação"; Sobre os exemplos de código apresentados nos Livros de Receitas, os sujeitos do grupo **Receitas** comentaram que estas os ajudaram na solução ou guiaram para os exemplos de código no sistema, veja as respostas: *"Foi útil para saber como instanciar o objeto Tool"*; *"Os exemplos facilitam o entendimento dos hot-spots"*; *"Atraves dos exemplos foi possível ir direto na classe que pudesse ser usada/aproveitada para fazer a atividade"*; *"Analisando a classe LineConnection, eu descobri a classe ArrowTip que ajudou na implementação de uma nova classe para o pentágono"*.

As respostas levam a acreditar que ambas as documentações foram capazes de guiar os sujeitos para os exemplos de código fonte. Porem o Livro de Receitas fez isso de maneira que gerou maior satisfação e percepção de facilidade de localização das informações e melhor navegação no documento. Também, se tem a percepção que os profissionais depois que identificam um código exemplo, preferiram usá-lo. Mas, e se não houver exemplo disponível e se este exemplo não for facilmente localizado, como para o JHotDraw e JModeller, por se tratarem de sistemas e *framework* pequenos. Ou, os sujeitos forem iniciantes e tiverem dificuldade em analisar o código para localizar exemplos como no caso dos experimentos com alunos da Instituição 2, que também utilizou o *framework* JHotDraw. Talvéz o direcionamento dado pelos documentos seja mais útil nestes casos. A utilidade das informações foi significamente maior para o grupo Receitas nos experimentos com alunos da Instituição 1 e Instituição 2.

5.3.4 Conclusões dos Resultados

O **tempo de execução** não apresentou significante diferença em nenhum experimento. Apesar de no experimento com alunos da Instituição 1 ter apresentado p-value 0,07, (Tabela 10) próximo ao nível de significância e tendendo a ser melhor para o grupo Receitas.A **taxa de acerto** foi significantemente melhor para o grupo Receitas no experimento com alunos da Instituição 1. Para os demais experimentos apresentou um resultado sem significante diferença entre os grupos. No experimento com alunos da Instituição 2, devido ao nível de experiência dos alunos, a atividade de instanciação foi considerada complexa pelos sujeitos dos dois grupos. Neste experimento, 93,33% dos sujeitos gastaram os 50 minutos de atividade e apenas 1 sujeito concluiu 100% da atividade. Para o experimento com profissionais, o tempo não foi delimitado e também não apresentou diferença significante entre os grupos de sujeitos. Estes, em sua maioria, terminaram a atividade com sucesso. Mas, muitas das respostas abertas deste experimento indicam que a maioria dos sujeitos de ambos os grupos preferiram o uso dos exemplos de código que estavam disponíveis para todos no próprio código fonte da aplicação a ser alterada e nos exemplos de sistemas que acompanham o *framework*. É importante observar que para o experimento com alunos da Instituição 1, foi passada apenas uma classe incompleta para cada grupo e os usuários não tiveram como consultar códigos em sistemas, além dos exemplos de códigos apresentados pelas documentações. Logo, as atividades do experimento

alunos da Instituição 1 foram apoiadas unicamente pelos documentos.

Sobre a **utilidade das informações**, estas foram significantemente úteis para os grupos com alunos da Instituição 1 e Instituição 2. Não houve significante diferença entre os grupos para o experimento com profissionais, onde a maioria dos sujeitos de ambos os grupos preferiu utilizar os exemplos de código presentes no sistema e nos exemplos do *framework*. Nos experimentos com alunos da Instituição 1, os documentos foram o único suporte para realizar as atividades. Para os alunos da Instituição 2, supõe-se que a falta de experiência com o IDE e com a análise de códigos os levou a depender mais da documentação para conseguir resolver as atividades. As informações do livro de receitas se diferenciam por serem estruturadas por características que podem ser implementas, apresentando os *hot-spots* que podem ser utilizados acompanhados de exemplos de código que apresentam como estes foram utilizados para aquela característica em outras aplicações. Exemplos foram relatados pelos participantes como sendo úteis, principalmente pelos mais experientes.

Sobre a **complexidade das atividades** percebida pelos sujeitos, no experimento com alunos da Instituição 1, o grupo Receitas apresentou uma significante percepção de menor complexidade das atividades. O Livro de Receitas pode ter auxiliado na execução das atividades, levando a uma maior taxa de acerto, maior percepção de utilidade do documento e consequente menor percepção da complexidade da atividade. Nos demais experimentos não houve significante diferença entre os grupos. No experimento com alunos da Instituição 2, dado ao nível de experiência dos sujeitos, estas foram avaliadas como complexas pelos sujeitos dos dois grupos. No experimento com Profissionais, a maioria dos sujeitos acabaram buscando por exemplos de código e em ambos os grupos esta foi a mesma base para realizar as atividades. Esta avaliação de complexidade pode ser um reflexo das taxas de acerto e tempos de execução não terem apresentado diferenças significantes neste experimento com profissionais.

A **facilidade de localização das informações** foi significantemente maior para o grupo Receitas nos experimentos com alunos da Instituição 2 e com Profissionais (*framework* JHotDraw). O experimento com alunos da Instituição 1 (*framework* JFace) não apresentou resultado significantemente diferente. A facilidade de localização das informações é influenciada pela estrutura da documentação e pela presença da informação necessária. A facilidade de localização das informações é influenciada por esta informação existir no livro de receitas, que está obviamente relacionada com a disponibilidade e qualidade de exemplos de instanciações de características que possam ser usados para gerá-lo. Porém, os uso cada vez mais frequente de repositórios de software tende a viabilizar a abordagem proposta. A localização das informações no livro de receitas é dada pela sua estrutura organizada por receitas para as características que podem ser instanciadas e apresentam os *hot-spots* que podem ser utilizados. As mesmas documentações foram utilizadas nos experimentos com alunos da Instituição 2 e Profissionais, o que oferece

uma opinião mais ampla e permite dizer que o livro de receitas conseguiu ser melhor para a localização das informações do que o documento tradicional do JHotDraw. No documento tradicional do JHotDraw, as informações sobre os *hot-spots* foram apresentadas por pacotes, isso dificulta associá-los as características que se deseja instanciar.

O fato do livro de receitas não apresentar uma significante diferença em relação a facilidade de localização da documentação tradicional do JFace é satisfatório, pois esta é uma documentação bem estruturada, onde as características importantes para as atividades estavam apresentadas separadas por tópicos, com artifícios de navegação e localização. Uma observação importante, é que mesmo para os experimentos com sujeitos Profissionais, onde se preferiu utilizar mais o próprio código fonte, este resultado e os relatos mostram que as informações presentes no código, e principalmente nos exemplos presentes nos documentos, os ajudaram a localizar os exemplos de código utilizados e os ajudou a entender o uso de certos *hot-spots*. Veja este relato: *"O processo de descoberta da solução foi mais orientado pela análise do código e busca de implementação por analogia, do que pela documentação em si. A documentação serviu, particularmente, para me mostrar como instanciar um objeto Tool conforme minha necessidade"*.

Sobre a **utilidade da navegação**, em todos os experimentos o Livro de Receitas obteve uma avaliação de utilidade da navegação significantemente maior do que as documentações tradicionais. Como falado anteriormente, no Livro de Receitas, as informações estão agregadas pelo interesse inicial do desenvolvedor que é a característica ligada ao que ele deseja desenvolver. Isto somente é possível por conta da análise dinâmica das características que permite localizar os *hot-spots* e outros elementos de código nos exemplos. Pela avaliação dos sujeitos, esta estrutura teve uma significante maior utilidade e pode ter influenciado nos resultados da **facilidade de localização** das informações nos documentos para 2 dos 3 experimentos.

A **clareza das informações** apresentada nos documentos apresentou um resultado melhor para o grupo Receitas no experimento com alunos da Instituição 2, e nos demais experimentos não apresentou um resultado significantemente diferente entre os grupos. Isto pode ser considerado satisfatório para uma abordagem gerada semi-automaticamente. Esta variável não seguiu uma tendência e não recebeu comentários dos sujeitos que ajudassem na avaliação dos possíveis motivos deste resultado.

Sobre a **satisfação com o uso dos documentos**, nos experimentos com alunos da Instituição 2 e com Profissionais, os resultados mostram significante maior satisfação em utilizar o livro de receitas. No experimento com alunos da Instituição 1, o p-value foi 0,1655 (Tabela 10) não permitindo dizer que foi significante maior para o grupo Receitas. Porém, no boxplot da Figura 26, observa-se que a mediana e primeiro quartil do boxplot do grupo Receitas é superior à mediana e primeiro quartil no boxplot do grupo Controle, indicando uma tendência de percepção maior de satisfação quando utilizando o livro de receitas. A satisfação com o uso do documento é uma variável importante, pois pode

refletir a percepção de satisfação gerada pela utilidade das informações, facilidade de localização das informações, utilidade do mecanismo de navegação quando comparado ao outro documento que é oficial para os *frameworks* e apresenta informações geradas pelos especialistas.

A percepção de clareza das informações, utilidade da navegação dos documentos, facilidade de localização das informações, utilidade das informações e percepção de complexidade das atividades utilizando os Livros de Receitas foram melhores ou iguais ao uso dos documentos tradicionais, o que indica uma certa qualidade dos Livros de Receitas gerados semi-automaticamente para os dois *frameworks* objetos de estudo. Os documentos tradicionais frequentemente são gerados manualmente por especialista e complementado com algum Javadoc que é automatizado.

Outro resultado que deve ser considerado é **taxa de cobertura** dos livros de receitas, que foi 100% das classes e métodos *hot-spot* para o *framework* JHotDraw e 100% das classes *hot-spots* e mais de 95% dos métodos *hot-spots* para o *framework* JFace. Estas taxas foram obtidas comparando as informações apresentadas nos Livros de Receitas com os exemplos de implementação das características em sistemas que utilizam os *frameworks*. Estas informações foram obtidas utilizando engenharia reversa (análise dinâmica e análise estática) e através da aplicação de alguns filtros. A cobertura das informações se mostrou suficiente para garantir resultados tão bons quanto os dos documentos tradicionais para os experimentos realizados.

A **utilidade dos exemplos** de código foi confirmada pelos resultados ligados ao uso do Livro de Receitas, mas, principalmente pelo relato dos sujeitos, que comentaram a importância dos exemplos para execução das atividades. Além de os sujeitos profissionais, com maior experiência, indicarem uma preferência por utilizar exemplos de código ao invez de documentações durante a execução das atividades de instanciação, quando existem e facilmente localizam estes exemplos. Também, é importante ressaltar que os Livros de Receitas ajudaram na localização dos *hot-spots* e exemplos de códigos para estes profissionais que preferem utilizar o código fonte como base de aprendizado.

Contudo, para os 3 experimentos envolvendo ao todo 44 sujeitos humanos, com 88 execuções de atividades reais de instanciação de *frameworks*. Onde o uso de livros de receitas foram comparados ao uso de documentações tradicionais dos *frameworks* na realização de atividades reais de instanciação. Nestes, os Livros de Receitas gerados semi-automaticamente apresentaram resultados de uso iguais ou melhores, em termos de taxa de acerto, tempo de execução e percepção da satisfação dos usuários, cumprindo os objetivos da pesquisa.

5.4 Conclusão do Capítulo

Neste capítulo foi apresentado o delimento experimental de três experimentos com 44 sujeitos humanos, realizando atividades reais de instanciação de *frameworks* para comparar os Livros de Receitas propostos com os documentos tradicionais de dois *frameworks*. A fim de obter informações sobre o tempo de execução e a taxa de acerto durante a execução das atividades, e a satisfação de uso dos documentos. Os resultados destes experimentos foram apresentados e discutidos, evidenciando que o objetivo de demonstrar a viabilidade de uso de livro de receitas para a instanciação de *frameworks* construídos por meio de análise estática e dinâmica foi alcançado.

Capítulo 6

Trabalhos Relacionados

Na última década, uma grande variedade de técnicas de documentação têm sido propostas para apoiar a compreensão de *frameworks*. Os autores em (FAIRBANKS; GARLAN; SCHERLIS, 2006) categorizam os trabalhos que tratam o problema da compreensão de *frameworks* e os divide em dois tipos de abordagem: **(a)** as baseadas em documentação prévia (especificação) e **(b)** as baseadas em exemplos de instanciação. Apresenta ainda uma dimensão para descrever se as informações geradas pelas abordagens prescreve como os usuários devem usar o *framework* ou descreve a implementação do *framework*. Para o trabalho aqui apresentado, os principais interesses estão nas abordagens baseadas em exemplos e nas que prescrevem como usar o *framework*, uma vez que a base das informações que compõem o livro de receitas são extraídas de exemplos de instanciação e visa prescrever o uso do *framework*. Mas, as abordagens não baseadas em exemplos também contribuíram para este trabalho, pois tratam de trabalhos que prescrevem como o usuário deve usar o *framework*, formas de apresentação das informações, informações relevantes para a instanciação de *frameworks*, problemas presentes na instanciação, experimentos e resultados importantes para a elaboração da abordagem apresentada nesta tese. Trabalhos que tratam sobre os problemas da instanciação de *frameworks* também são trabalhos relacionados importantes para a definição da abordagem, especialmente no quesito definição do conteúdo e organização do livro de receitas. A seguir serão apresentados os trabalhos considerados de maior importância pela autora para a definição desta tese.

6.1 Problemas da Instanciação de *Frameworks*

A seguir serão apresentados trabalhos que tratam os problemas que ocorrem durante o processo de instanciação de *frameworks* e que podem ser tratados com uma documentação adequada.

O que exatamente faz uma API difícil de aprender? Essa é a pergunta principal do trabalho de Robillard 2009. Para responder a essa pergunta, ele investigou os obstáculos que os desenvolvedores profissionais da Microsoft enfrentavam quando aprendiam a usar

APIs, usando uma abordagem completamente fundamentada na experiência dos desenvolvedores. Um total de 83 desenvolvedores responderam à pesquisa. Para focar nas respostas da pesquisa e fazer com que os desenvolvedores pensem sobre especificidades, Robillard lhes pediu que comentassem suas experiências de aprendizado de APIs mais recentes.

Os desenvolvedores foram questionados sobre: Quais obstáculos dificultaram a aprendizagem da API? As respostas sobre as estratégias de aprendizagem produziram poucas surpresas. Dos 80 entrevistados, 78% deles aprenderam APIs lendo documentação, 55% usaram exemplos de código, 34% experimentaram APIs, 30% leram artigos e 29% perguntaram a colegas. Os itens de menor freqüência incluíam uma grande variedade de outras estratégias, como ler livros ou rastrear código através de um depurador. Além de fornecer uma visão geral de como os desenvolvedores tem o primeiro contato com as APIs, esta parte da pesquisa ofereceu poucas oportunidades para uma análise mais aprofundada. O restante do artigo está preocupado com os obstáculos que os desenvolvedores enfrentam quando aprendem APIs.

Um grande resultado da pesquisa é que os recursos de aprendizagem superaram a lista de obstáculos para aprender APIs. O autor lembra que os esforços para melhorar a usabilidade da estrutura de uma API precisam ser complementados por esforços para melhorar os recursos disponíveis para aprendê-los. Com exceção das queixas gerais sobre a documentação oficial divulgada com as API, essa classificação revela a variedade de desafios enfrentados pelo pessoal encarregado de produzir recursos de aprendizagem para APIs. Ou seja, para mitigar os obstáculos, a documentação da API deve: i) incluir bons exemplos; ii) estar completo; iii) suporte a muitos cenários de uso complexos; iv) ser convenientemente organizados; e, v) incluir elementos de projeto relevantes.

Pode-se observar a necessidade de compreender os aspectos de projeto e racionalidade da API de acordo com a necessidade, os obstáculos relacionados ao uso de exemplos de código e os desafios de lidar com o comportamento aparentemente inexplicável de uma API. Sete participantes disseram que para entender adequadamente uma API, eles precisavam entender seu projeto de alto nível. Exemplos podem tornar-se mais um obstáculo do que um recurso quando há uma clara incompatibilidade entre o propósito do exemplo e o objetivo do usuário. A maioria dos problemas com exemplos relacionava-se com participantes que desejavam usar pequenos exemplos para propósitos que iam além da interação básica com a API. Quando discutiam o uso dos exemplos, os participantes apontam que estes devem: i) oferecer "melhores práticas"de uso de uma API; ii) informar o projeto do código que usa a API; iii) fornecer justificativa sobre o projeto de uma API; e, iv) confirmar as hipóteses dos desenvolvedores sobre como as coisas funcionam.

Entre estes pontos, a frustração mais frequente foi que pequenos exemplos de código [1] não fornecem qualquer apoio para pensar em "como juntar as coisas "(juntar as partes

[1] do inglês *snippets*

desenvolvidas para formar um sistema). Nos casos em que fragmentos não suportavam o objetivo do desenvolvedor. Outro ponto observado é que os exemplos que estão ligados aos criadores da API parecem mais atraentes, sendo esta considerada pelos desenvolvedores como a melhor prática. O autor argumenta sobre esta consideração lembrando que por mais razoável que seja essa suposição, deve ser feita com cuidado, porque as pessoas que escrevem a documentação da API na Microsoft não são as pessoas que desenvolvem a API.

O autor ainda complementa que ferramentas de software podem ajudar os desenvolvedores em sua busca por uma melhor compreensão das APIs. As ferramentas de busca são promissoras nessa área porque ajudam a preencher a lacuna entre as necessidades de informações dos usuários da API e os recursos correspondentes (como exemplos de código). À medida que as APIs continuam crescendo, os desenvolvedores precisarão aprender uma fração proporcionalmente menor do todo. Em tais situações, a maneira de promover experiências de aprendizado de API mais eficientes é incluir meios mais sofisticados para os desenvolvedores identificarem a informação e os recursos de que necessitam, mesmo para APIs bem projetadas e documentadas.

Este trabalho de Robillard ajuda na justificativa da importância de abordagens e ferramental como os apresentados nesta tese. Esta abordagem contribui para a documentação de *frameworks*, que tem por base uma API. Os Livros de Receitas apresentam exemplos de código que são considerados importantes para o aprendizado de APIs. As informações e exemplos que compõem o Livro de Receitas são retirados de soluções completas para as características ligadas a necessidade do usuário, apresentando assim exemplos completos de uso da API. Além do mais, cada receita trata de uma característica que pode ser desenvolvida utilizando o *framework*, para poder apresentar apenas a fração de informações que interessa ao desenvolvedor. Padrões de projeto foram inseridos nos Livros de Receitas desta tese pensando na necessidade de apresentar informações sobre o projeto que deve ser seguido. Informações sobre o projeto para o desenvolvimento foram apontados como informações muito importantes para o aprendizado da API.

Em (KIRK; ROPER; WOOD, 2005), os autores afirmam há pouca pesquisa para identificar os problemas específicos que possam surgir durante a reutilização de *framework* e avaliar as técnicas de documentação em relação a estes problemas. Então, os autores realizaram um primeiro estudo com o *framework* JHotDraw para identificar tais problemas de instanciação dos *frameworks*. A análise revelou quatro grandes categorias de problemas: 1) **Mapeamento** (38 problemas), identifica o problema de traduzir uma solução abstrata, conceitual, para uma implementação concreta que reutiliza as estruturas existentes do *framework*; 2) **Compreender as funcionalidades** (60 problemas), descreve problemas de compreensão sobre o que realmente faz cada parte específica do *framework*; 3) **Compreender as interações** (48 problemas), concentra-se em problemas relacionados com a comunicação entre classes do *framework*; 4) **Compreender a arquitetura** do

framework (17 problemas), é o problema de fazer modificações sem dar a devida atenção às qualidades arquitetônicas de alto nível do *framework*.

A segunda fonte de dados foi capturado com as experiências de quatro estudantes de último ano de um projeto chamado *Honours*. Cada aluno foi atribuído um problema individual para resolver usando o *framework*. Projetos duraram um período de aproximadamente seis meses e envolveu os alunos em todas as áreas do ciclo de vida de desenvolvimento de software. A terceira fonte de dados foi coletada a partir de uma classe de graduação de Arquitetura de Software. Perguntas foram feitas sobre a gama de problemas experimentados e o efeito das documentações sobre esses problemas. Este é um bom mecanismo para consultar uma grande população (70 alunos), mas ele pode sofrer uma taxas de resposta pequena (neste caso, apenas 18 estudantes responderam). Um total de 209 problemas relacionados à *frameworks* foram capturado por esta investigação, 59 destes problemas foram derivados a partir do estudo individual, 35 a partir do projeto com estudantes e 115 problemas a partir da classe de Arquitetura.

Uma variedade de técnicas e estratégias de documentação de *framework* foram revistas e *pattern languages* foi escolhida para abordar o problema do mapeamento e micro arquiteturas. Este estudo avaliou as dificuldades: mapeamento, interação, funcionalidade e arquitetura versus o acesso e apoio das documentações: *pattern language*, micro arquitetura, código fonte e conhecimento prévio. *Pattern languages* para a solução de problemas de projeto foram originalmente proposto pelo arquiteto ALEXANDER et al. (1977) no contexto da arquitetura. *Pattern languages* apresenta uma decomposição do projeto em uma coleção de subproblemas que podem ser abordados de forma independente do todo. Cada sub-problema é descrito por um padrão que identifica as forças que devem ser consideradas na resolução do subproblema e propõe uma solução que resolve estas forças. A linguagem é formado pelas relações entre esses padrões. O estilo dos padrões permaneceu essencialmente como em um livro de receitas (KRASNER; POPE, 1988), mas mais exemplos foram incluídos para tentar melhorar sua profundidade técnica.

Como resultado do estudo, os autores concluíram que o mapeamento dos problemas ainda parece ser o mais importante dos problemas detectados. Os autores atribuíram atividades de reutilização por um longo período de tempo, por sua natureza, outros problemas podem ocorrer ou serem evitados dependendo de como a solução foi mapeada para as partes existentes do *framework*. Problemas de interação foram encontrados, e ocorrerem com menos frequência do que inicialmente previsto. Em parte, isso pode ser por causa da situação experimental, que não foi ao nível de detalhes de codificação da solução. Mas, também pode refletir que este tipo de conhecimento é necessário com menor frequência. Problemas de Arquitetura não foram bem capturados nos estudo, pois eles exigem uma escala de tempo mais longa para ter efeito.

Há evidências de que *pattern languagens* fornecem algum suporte para os problemas de mapeamento, especialmente para os usuários sem experiência com o *framework*, através

da introdução de conceitos chave e fornecendo exemplos de utilização do *framework*. No entanto, ficou claro que a experiência anterior dominava o uso explícito da *pattern language*. Para sujeitos experientes, alguma dessa experiência foi adquirida com o uso prévio da *pattern language*. Além disso, o uso extensivo de experiência anterior pode ser um provável inibidor de outras formas de documentação, com seu imediatismo muitas vezes impede a consideração de soluções alternativas. A microarquitetura foi concebida para ajudar a desenvolver na compreensão das interações fundamentais do *framework*, mas os resultados sugerem que era relativamente ineficaz. Problemas particulares com o estilo de micro arquitetura usada, levaram os participantes à terem dificuldades em identificar qual a interface do *framework* deveria ser usada como uma rota para a microarquiteturas. Participantes não juntaram as partes para compreender as unidades maiores de interação subjacentes ao *framework*. Embora as microarquiteturas utilizadas no estudo foram relativamente ineficazes, os autores ainda acreditam que a documentação desta natureza é necessário para resolver os problemas de interação.

Este trabalho (KIRK; ROPER; WOOD, 2005) contribuiu para o trabalho apresentado nesta tese, ao destacar a importância do mapeamento, interações, funcionalidades e arquitetura versus o acesso e apoio das documentações. O que influenciou na definição da estrutura e conteúdo do Livro de Receitas apresentado nesta tese.

O trabalho (KIRK; ROPER; WOOD, 2005) apresenta uma avaliação sistemática dos principais problemas relacionados à instanciação de *frameworks*. Este ajudou na análise dos problemas que a presente proposta pretende abordar e elucidou sobre a contribuição de cada tipo de informação. O estudo com sujeitos humanos apresentado neste artigo será de grande valia na definição dos experimentos desta tese, uma vez que também realizaram experimentos com sujeitos humanos. Outro ponto que chamou a atenção foi que os experimentos realizados pelos autores não chegaram ao nível de detalhes de codificação da solução, o que eles assumem poder influenciar na não avaliação correta da necessidade da documentação de interação. Nos experimentos apresentados nesta tese, realizou-se estudos com atividades que concretizam com a implementação, codificação, de uma solução.

Em outro trabalho (HOU; WONG; HOOVER, 2005), os autores afirmam que para tornar os *frameworks* mais fácil de usar, precisamos entender melhor as dificuldades que os programadores têm com eles. As perguntas que os programadores fazem dão pistas sobre a qualidade do projeto, documentação e práticas do programador. Este artigo descreve o método e os resultados de um estudo sobre o Java Swing *Framework*, onde os autores coletaram e analisaram uma amostra de 300 perguntas sobre dois componentes do *Swing* (*JButton* e *JTree*), e classificaram as perguntas de acordo com as características de projeto dos componentes. Este processo revelou informações importantes que podem melhorar o projeto de um *framework*, os tutoriais e práticas do programador.

Este estudo explora a forma como a informação implícita em perguntas feitas pelos

programadores em fóruns pode sugerir melhorias para projeção, documentação e prática de programação. Os autores se concentraram no *Java Swing Framework* porque este é maduro, bem documentado e amplamente utilizado. O estudo, manualmente, recolheu e classificou as perguntas do grupo de notícias do Swing Forum. Eles escolheram dois componentes do Swing: JButton e JTree. Jbutton é um componente representativo e simples, que revela as questões relacionadas com a herança profunda. JTree representa um componente tipicamente complexo com uma composição rica. Os autores seguiram o seguinte processo neste estudo: **1)** identificar as características do projeto para o objeto de estudo; **2)** coletar questões relevantes; **3)** analisar e classificar cada pergunta pela característica que é a causa raiz do problema.

As seguintes conclusões foram apresentadas. Os autores acreditam que a documentação agrupada logicamente por característica torna mais fácil o aprendizado sobre o *framework* para os programadores, pois estes ajudam os programadores evitarem grande parte do que é desnecessário para a solução. Apesar da alta qualidade do tutorial, estes não são perfeitos. A maioria dos **programadores não estudam** sistematicamente o projeto de um *framework*. Um bom projeto e documentação deve tornar o *framework* mais fácil usar. Os programadores, por vezes, precisam ser informados quando a tarefa que requer um esforço sério para a obtenção de uma compreensão mais profunda. **Pontos de variação intimamente ligados**. Um ponto de variação (*hot-spots*) fornece aos programadores a capacidade de fazer o reúso do *framework*. Pode ocorrer de um ponto de variação depender de muitas partes do *framework* , considera-se fortemente acoplado. Neste último caso, os projetistas do *framework* também poderiam ajudar, antecipando típicos cenários de uso e fornecendo implementações padrão. **Interesses não localizados**. Um design orientado a objetos tais como JTree muitas vezes consiste em um número considerável de classes e interfaces, e o código-fonte de uma característica ou interesse [2] podem estar espalhados em várias construções sintáticas. Como modularizar tais preocupações é um tópico de pesquisa interessante. Mas para os profissionais, uma solução prática é documentar esses casos com cuidado, para manutenção e reutilização futura. **Conselho: remover casos especiais sempre que possível.** Um conselho para os projetistas é remover casos especiais dos *frameworks* sempre que possível. Universalmente propriedades mantidas têm a característica de serem aprendidas uma vez e aplicadas por toda parte. Casos especiais forçam os programadores a aprenderem informações extra, e isto impede a compreensão e aumenta a chance de cometer erros. Fóruns desempenham um papel significativo em ajudar os programadores na tarefa de aprendizagem, e informam os desenvolvedores do *framework* sobre problemas no projeto ou documentação. Primeiro, os fóruns, em geral, podem ser excelentes recursos para a depuração, no aprendizado sobre erros de plataforma, e na discussão sobre questões de design. A segunda vantagem do fórum é que são úteis para descobrir rapidamente se um problema no programa é devido a

[2] do inglês concern

um *bug* de plataforma. As pessoas também trazem seus problemas de *design* para o fórum e os discutem com os seus pares, o que em muitas vezes termina com boas sugestões.

Este trabalho é importantes para a elaboração da presente proposta, pois apresenta pontos de necessidade de melhoria, dificuldades identificadas e possíveis soluções para o processo de reuso de *framework*, sendo este o problema tratado pelo trabalho proposto e suas conclusões influenciaram a definição da abordagem. Este reforça a necessidade de dar foco nas documentações para casos específicos de uso, como características importantes do *framework*, que é parte da solução que propomos nesta tese. Outro ponto importante que tratamos na abordagem aqui proposta é tratar o problema do interesse (características) não localizado.

6.2 Abordagens Baseadas em Especificação

A seguir serão apresentados com mais detalhes os trabalhos relacionados baseados em especificação prévia ou conhecimento de especialista.

Em (ORTIGOSA; CAMPO, 1999), os autores afirmam que os chamados livros de receitas ativos [3] representam um dos mais proeminentes exemplos de ferramentas que fornecem assistência semi automatizada para o processo de instanciação do *framework*. Livros de receitas ativos são capazes de apresentar descrições de receitas, proporcionando ao usuário uma interface interativa para guiá-lo através do processo de instanciação. Os autores apresentam os *SmartBooks*, uma nova abordagem para apoiar a instanciação do *framework* baseado no conceito de livro de receitas ativo. A ferramenta apresenta ao desenvolvedor as diferentes atividades de alto nível que podem ser realizadas na criação de uma nova aplicação com o *framework*, tomando como base a documentação fornecida pelo projetista por meio de regras de instanciação. Para cada uma das atividades de alto nível, há uma lista de tarefas que o usuário deve seguir a fim de completar a atividade, esta lista de tarefas é chamada plano de instanciação.

Autores afirmam que uma ferramenta mais poderosa deve fornecer ao usuário um mecanismo para expressar os requisitos que a sua aplicação específica deve cumprir, e fornecer orientações sobre quais atividades de programação devem ser realizadas a fim de obter tais comportamentos com o *framework*. A Figura 35 apresenta exemplos de regras específicas de um *Smartbook* para p *framework* HotDraw de editores gráficos (JOHNSON, 1992).

O *SmartBooks* é dividido em seis componentes principais, apresentados a seguir. **Ferramenta de Documentação** (Documentation Tool): esta é usada principalmente pelo desenvolvedor do *framework* para escrever a documentação deste. **Gerador de Regra** (*Rule Generator*): este componente gera as regras que são usadas na geração do plano de instanciação com base na descrição do *framework*. **Coletor de Funcionalidade**

[3] do inglês active cookbooks

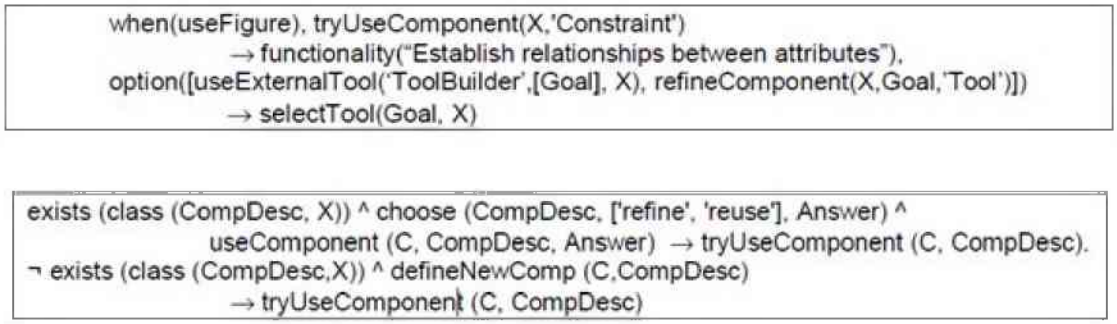


Figura 35 – Exemplo de regras específicas de um *framework* de editores gráficos, que usam o *framework* HotDraw (ORTIGOSA; CAMPO, 1999)

(*Functionality Collector*): usando informações sobre as funcionalidades do *framework*, este módulo ajuda o usuário na descrição da funcionalidade requerida para a aplicação em desenvolvimento. **Planejador** (*Planner*): com base nas informações de requisitos e nas regras, este gera um plano parcial para criar a aplicação por meio de instanciação do *framework*. O plano é gerado utilizando um algoritmo de planejamento chamado PIT [8]. **Gerenciador de Tarefas** (*Task Manager*): controla a atividade do desenvolvedor do aplicativo. Este gerencia a lista de tarefas pendentes e quando uma tarefa é concluída, para que se possa verificar se alguma regra de consistência deve ser aplicada. **Gerenciador de regra** (*Rule Manager*): seu trabalho está relacionado com o Gerenciador de Tarefas. O Gerenciador de regra tem a responsabilidade de verificar se as ações do usuário produzem uma configuração de software inconsistente com a descrição do *framework*. Nesse caso, este deve criar e passar, para o Gerenciador de tarefas, as tarefas que o usuário tem de executar para voltar a uma configuração consistente.

A principal contribuição de (ORTIGOSA; CAMPO, 1999) é mostrar como o processo de instanciação pode ser representado como uma sequência de tarefas do usuário, que são obtidas por um algoritmo de planejamento a partir da descrição funcional do aplicativo. Uma das limitações da abordagem proposta é a necessidade de descrever a funcionalidade implementada pelo *framework* utilizando linguagem natural. Vocabulário, suposições e visões do mundo podem ter grandes variações de um usuário para outro, e, especialmente, com relação ao projetista do *framework*. Devido a isso, expressar a funcionalidade e compreender as capacidades do *framework* são tarefas difíceis impostas ao projetista e usuários, respectivamente. Este problema deve ser cuidadosamente estudado, de modo a encontrar uma maneira menos ambígua de descrever a funcionalidade, sem as dificuldades do uso de alguma notação formal.

Ortigosa e Campo (1999), apresentam uma abordagem sistemática para obtenção do *Smartbook*. Os autores apresentam uma proposta de linguagem para a definição de regras, lista de tarefas, receitas, sendo este trabalho importante para a definição da estrutura e informações presentes no Livro de Receitas apresentado nesta tese. Como diferencial, o trabalho apresentado nesta tese gerar as receitas a partir de engenharia reversa e não a

partir do conhecimento de especialista ou documentação como ocorre nos artigos desta seção.

Outro trabalho (OLIVEIRA et al., 2004), apresenta uma abordagem para a instanciação do *framework* baseado em processos de software que permite o desenvolvimento da aplicação final de uma forma sistemática. A abordagem compreende um conjunto de técnicas de representação e análise de estrutura que inclui uma representação do projeto do *framework* para extensão baseado em uma extensão da UML (*Unified Modeling Language*) (BOOCH; RUMBAUGH; JACOBSON, 2005) e uma linguagem especificação baseada em processos de software. Esta abordagem indica **como** a instanciação deve ser realizada, **quais** atividades de instanciação devem ser realizadas, e **quando** essas atividades são realizadas para que as restrições do sistema sejam atendidos.

Para alcançar o objetivo e resolver os problemas relacionados com a instanciação do *framework* anteriormente mencionado, os autores: 1) desenvolveram UML-FI (*UML Framework Instantiation*), uma extensão da UML, que permite a representação de pontos de extensão do *framework* e 2) definiram a RDL (*Reuse Description Language*), uma linguagem de processo para a representação de atividades de instanciação do *framework*. Eles também definiram técnicas que permitem a validação das extensões do *framework*. Eles ilustram a abordagem com o JUnit, um *framework* simples, mas útil para testar programas.

Uma abordagem sistemática deve ser seguida durante a instanciação do *framework*, que consiste nas seguintes etapas principais: **1)** extensão do projeto do *framework*. Esta etapa consiste em definir um projeto estendido usando UML-FI, que inclui os pontos de extensão, onde o *framework* deve ser adaptado; **2)** representação da instanciação do *framework*. Nesta etapa, a sequência de tarefas necessárias para produzir as instâncias são definidas usando RDL; **3)** análise da instanciação do *framework*. Utilizando a ferramenta de instanciação é possível validar as tarefas instanciação do *framework* no sentido de que as instâncias não violam as propriedades estruturais, tais como as regularidades estruturais e boa formação.

A fim de organizar declarações RDL, os autores adotaram o conceito de livro de receitas. As receitas podem ser rastreadas para funções em uma linguagem de programação imperativa que apresenta o próprio código de instanciação. É importante notar que cada livro de receitas deve ter uma receita principal (chamado de *main*) que especifica o ponto de entrada de execução. Os pontos de extensão são espaços reservados que indicam onde um *framework* deve ser adaptado. Sua abordagem centra-se na instanciação de *frameworks* orientados a objetos, por isso deve ser capazes de expressar a natureza do ponto de extensão (os *hot-spots*) e as atividades instanciação relacionadas em termos de técnicas de programação orientada à objetos (OO).

Os autores deste artigo tratam o mesmo problema alvo da presente proposta com o mesmo foco nos elementos de instanciação do *framework*. Este é um trabalho bem

estruturado e com uma solução completa para atender a instanciação de *frameworks*. Este ajudou no processos de definição da abordagem apresentada nesta tese, por ser uma bom conjunto ferramental e estrutura de apresentação das informações presentes no livro de receitas. O principal diferencial, conforme já apresentado, é que o trabalho aqui proposto pretende não depender de documentação ou projetista para recuperar as informações necessárias para a instanciação, fazendo isto por meio de engenharia reversa.

O trabalho de FAIRBANKS et al.(2006) descreve duas contribuições para o problema tratado nesta tese. Em primeiro lugar, eles fornecem uma técnica melhorada para ajudar os programadores a superar os encargos do uso de *frameworks*. Eles descrevem os conceitos e a linguagem usada para expressar os *design fragments*, além das ferramentas que eles construíram para permitir verificar a conformidade entre o código de programadores e o *design fragment*. Para a descrição do que o programador deve fazer, eles adicionaram uma descrição minimamente suficiente de como o *framework* vai agir sobre o código do programador para que o programador possa racionalizar sobre a interação do código com o *framework*.

Um *design fragments* é um padrão que codifica uma solução convencional de como um programa interage com um *framework* para alcançar uma meta. Este tem duas partes principais. A primeira é uma descrição do que o programador deve construir para cumprir a meta deste *design fragments*, incluindo as classes, métodos e campos que devem estar presentes. Esta descrição também inclui o comportamento destes métodos. A segunda parte do *design fragments* é uma descrição das partes correspondentes do *framework* que interagem com o código do programador, incluindo os métodos de retorno de chamada que serão chamados, os métodos do serviço que são providos, e outras classes do *framework* que são utilizadas.

Um *design fragments* oferece ao programador ajuda para compreensão do *framework* com uma "lanterna inteligente". Esta lanterna inteligente ilumina apenas as partes do *framework* que ele precisa entender para a tarefa em mãos. Sem a lanterna inteligente, um programador "navega" nas classes do *framework* e é "inundado" com detalhes de implementação privadas ou incapazes de diferenciar o relevante do irrelevante. Design fragments tem dois benefícios imediatos para os programadores. Em primeiro lugar, as ferramentas de análise pode verificar a conformidade entre a intenção declarada do programador e seu código-fonte. Em segundo lugar, os programadores que não conhecem uma parte do *framework* podem rapidamente encontrar uma solução no catálogo. Os autores acreditam que isto é essencial para fornecer aos programadores um valor imediato para o investimento de esforço.

Sobre a linguagem do *design fragments*, a intenção da linguagem é a de expressar: a estrutura do código do programador , os requisitos comportamentais do código do programador; a estrutura relevante do código do *framework* e o comportamento relevante do *framework*. A Figura 36 apresenta um exemplo de especificação utilizando esta linguagem,

```
<class name="java.applet.Applet"
    provided="yes">
 <method name="start" returnvalue="void">
  <freeformspec text="Callback method;
    invoked when framework decides to
    initialize your applet." />
  <invocation-cardinality value="1" />
  <invocation-lifecycle value="yes" />
  <invocation-type value="callback" />
  <invocation-pair value="stop" />
  <invoked-before value="stop" />
 </method>
...
</class>
```

Figura 36 – Exemplo da linguagem de d*esign fragments* (FAIRBANKS; GARLAN; SCHERLIS, 2006).

apresenta o método *start()* da classe *java.applet.Applet*.

A segunda contribuição é um estudo de caso detalhado sobre o *applet framework* que examina como cinquenta e seis *applets* usam *design fragments*. Os autores analisaram 20 *applets* demonstrativos oferecidos pela Sun e criou um catálogo representativo de *design fragments* da melhor prática convencional. Ao avaliar 36 *applets*, eles mostraram que *design fragments* são comuns, muitos applets copiaram a estrutura das demonstrações da Sun, e que a criação de um catálogo de *design fragments* é prático. *Design fragments* dão aos programadores benefício imediato por meio da garantia de conformidade com base em ferramentas e benefícios em longo prazo através da expressão da intenção do projeto.

O **estudo de caso** permite avaliar as seguintes hipóteses: **1)** o código-fonte que usa um *framework* é provável que siga padrões estruturais; **2)** programadores referem-se a exemplos para aprender a utilizar os *frameworks*; **3)** o esforço para criar um catálogo de design fragments vai diminuindo gradualmente e **4)** a linguagem de *design fragments* pode ser usada para expressar os padrões descobertos no código de exemplo.

Os resultados do estudo de caso apoiam as suas hipóteses apresentadas. Programadores instanciaram *applet framework* de maneira muito similar ao proposto. Com base nos *applets* dos programadores, tem a impressão que os programadores copiaram a estrutura das demonstrações da Sun. Os autores foram capazes de criar um catálogo rapidamente. E, com algumas exceções, a linguagem dos *design fragments* foi capaz de expressar os padrões encontrados. Este trabalho apoiou as hipóteses de que o código que interagem com os *frameworks* seguem padrões, programadores copiam exemplo de código, o esforço para criar um catálogo de *design fragments* diminui gradativamente, e que a linguagem *design fragments* é eficaz em expressar a maioria dos padrões observados.

Este trabalho ilumina apenas as partes do *framework* que o usuário precisa entender para a tarefa em mãos. A abordagem apresentada nesta tese também trata o problema por pedaços, por características, por necessidade, compreendendo que isto facilita

a compreensão do desenvolvedor. As hipoteses confirmada em (FAIRBANKS; GARLAN; SCHERLIS, 2006) são importantes para o trabalho apresentado nesta tese. Em especial a confirmação de que os exemplos de código auxiliam na instanciação de *frameworks*.

Em (OLIVEIRA; ALENCAR; COWAN, 2011), os autores aproveitam trabalhos anteriores dos autores sobre a definição da RDL, uma linguagem para facilitar a descrição do processo de instanciação, e descrever a ferramenta ReuseTool como uma maneira de ajudar no reuso de *frameworks* orientados a objetos. A ferramenta tem a finalidade de orquestrar as ações dentro de um processo de reutilização que é especificado pelo desenvolvedor do *framework* e executado de forma interativa pelo usuário do *framework*. O processo de reutilização é especificado usando a RDL, que é uma linguagem especial que permite a especificação de fluxos de processos, tais como sequenciamento, repetições e desvios, e também suporta comandos para manipular desenhos do *framework* baseados em UML. Como resultado, o ReuseTool é capaz de adaptar projetos para *framework* descritos em UML (2010) com novos elementos do modelo que representam as necessidades da aplicação em desenvolvimento. Este trabalho também descreve como a ferramenta pode ser estendida para incorporar novas atividades de reutilização e fornecer informações de seu uso com base em um estudo de caso exploratório.

Esta abordagem, por meio de um processo rigoroso pretende orientar o usuário do *framework* na extensão de todos os *hot-spots* necessários, em uma sequência predefinida pelo projetista do *framework*, seguindo assim a mesma lógica usada na implementação do *framework*. A fim de criar um ambiente de execução baseado em processo, os autores estabeleceram um conjunto de requisitos para a ferramenta. A ReuseTool visa facilitar a dimensão de Especialização com possíveis efeitos para Seleção e Integração. A dimensão Especialização para *frameworks* orientados a objetos lida com a adição de novos elementos de projeto e código para o desenho do *framework* original, onde cada novo elemento se conecta à um *hot-spots* do *framework*. A ligação entre o novo elemento de design e os *hot-spots* estende a estrutura com a informação que é específica para o aplicativo em desenvolvimento. Para orquestrar a especialização do *framework*, a ReuseTool utiliza um programa em RDL e o modelo UML do *Framework* como entrada e produz o modelo UML de aplicação (*Application UML Model*). O modelo UML do *framework* representa as classes e as relações, e também fornece informações úteis sobre como o código do *framework* pode ser obtido. O programa RDL detalha como o Modelo UML do *framework* deverá ser manipulado para acomodar novos elementos de projeto relacionados com a nova aplicação e o modelo UML da aplicação é o projeto final da aplicação. A Figura 37 apresenta um exemplo de receita em RDL.

O trabalho (OLIVEIRA; ALENCAR; COWAN, 2011) apresenta uma abordagem e ferramental que visam atender o mesmo problema do trabalho aqui proposto, sendo uma abordagem considerada bem estruturada pela autora desta tese. Dentre as linguagens avaliadas neste planejamento, a RDL foi considerada pela autora como uma das mais

```
1. COOKBOOK ShapesProducts
2. RECIPE main(
3.
4.      packA = NEW_PACKAGE(FrameworkModel,"org.reusetool.example.myshapeseditor");
5.      LOOP ("Create another shape?")
6.      {
7.          // PREPARE Images
8.          EXTERNAL_TASK("Define 16x16 icon");
9.          EXTERNAL_TASK("Define 24x24 icon");
10.
11.         // Define Shape SubClass
12.         shapeClass = CLASS_EXTENSION(Shape, packA,"?");
13.
14.         // Refine Abstract Methods
15.         m = METHOD_EXTENSION(Shape,shapeClass,addConnectionAnchor);
16.         ADD_CODE(shapeClass,m,"return new ChopboxAnchor(iFigure);");
17.         m = METHOD_EXTENSION(Shape,shapeClass,getFigure);
18.         ADD_CODE(shapeClass,m,"return  new IFIGURE_SUBCLASS();");
19.         m = METHOD_EXTENSION(Shape,shapeClass,getIcon);
20.         ADD_CODE(shapeClass,m,"return createImage(\"NEW_SHAPE.gif\");");
21.
22.         // Configure Palette
23.         ADD_CODE(ShapesEditorPaletteFactory,createShapesDrawer,
24.                         "component = new CombinedTemplateCreationEntry(
25.                 \"NEW_SHAPE\",
26.                 \"Create a NEW_SHAPE shape\",
27.                 NEW_SHAPE.class,
28.                 new SimpleFactory(NEW_SHAPE.class),
29.                 ImageDescriptor.createFromFile(ShapesPlugin.class, \"icons/crazy16.gif\"),
```

Figura 37 – Receita descrita utilizando a linguagem RDL (OLIVEIRA; ALENCAR; COWAN, 2011)

completas. Estas linguagens e a estrutura da receita gerada pela ReuseTool foram importantes para a definição da estrutura da Receita proposta para esta tese. Como diferencial, a abordagem proposta nesta tese pretende obter as receitas à partir de técnicas de engenharia reversa, enquanto na ReuseTool, a proposta da ferramenta é de orquestrar ações de reutilização dentro de um processo de reutilização que é especificado pelo desenvolvedor do *framework*.

6.3 Abordagens Baseadas em Exemplos

A seguir serão apresentados alguns dos trabalhos relacionados que tratam o problema da instanciação de *frameworks* e utilizam abordagens baseadas em exemplos, que obtem as informações de forma estática ou dinâmica a partir do código fonte de aplicações e/ou o *framework*.

Pesquisadores têm defendido a criação de repositórios para abrigar exemplos de uso de um *framework*, como em (NEAL, 1989), (YE; FISCHER; REEVES, 2000). Algumas abordagens dificultam o processo de localização e incorporaçãp dos exemplos a partir de repositórios. Pois, os desenvolvedores devem aprender uma nova linguagem de consulta como em (HENNINGER, 1991); ou, eles devem ter uma ideia de que tipo de exemplo provavelmente pode ajudá-los com as suas tarefas, como em (MICHAIL, 2000); ou, escrever o código fonte em um estilo que está em conformidade com a do exemplo presente no repositório, como em (YE; FISCHER; REEVES, 2000).

Para aliviar o fardo sobre o desenvolvedor, HOLMES e MURPH (2005) descreveram uma abordagem que utiliza a estrutura do código em desenvolvimento para encontrar exemplos relevantes em um repositório. A abordagem proposta em (HOLMES; MURPHY, 2005) tem duas vantagens. Em primeiro lugar, o contexto estrutural que é utilizado para formar uma consulta é extraído automaticamente a partir do código escrito pelo desenvolvedor. Um desenvolvedor que deseje procurar exemplos no repositório só precisa emitir um pedido de procura, usando uma combinação de teclas, para encontrar uma lista de exemplos relacionados. O desenvolvedor não precisa aprender uma nova linguagem de consulta, nem precisa escrever o código fonte usando normas específicas para permitir a consulta ao repositório. Em segundo lugar, o repositório de exemplos é extraído automaticamente a partir de aplicações existentes que usam o *framework*. Para investigar esta abordagem, os autores construíram a ferramenta Strathcona, um *plug-in* para o Eclipse (IDE) que extrai o contexto estrutural do código que o desenvolvedor está trabalhando para obter os exemplos com similaridade no repositório. Para obter os exemplos, o desenvolvedor deve escrever comentários que explicam a funcionalidade do software em termos semelhantes aos do repositório de código (YE; FISCHER; REEVES, 2000). As consultas feitas para o repositório são baseados nestes comentários e assinaturas de método. A parte do servidor da ferramenta abriga o repositório de exemplos e seleciona exemplos a serem devolvidos utilizando um conjunto de heurísticas para obter correspondentes estruturais. Nesta abordagem, um exemplo é um subconjunto de uma das aplicações presentes no repositório, que consiste em um conjunto de classes e relações relevantes.

Na avaliação desta abordagem, a questão-chave de interesse era saber se a heurística de correspondência estrutural pode retornar exemplos que um desenvolvedor considerava útil. Como só é possível compreender se um exemplo é útil no contexto de uma tarefa, foi realizada uma avaliação qualitativa, onde dois sujeitos replicaram quatro casos; cada caso consistiu de uma tarefa de programação relacionada a desenvolver *plug-ins* para o Eclipse. Os sujeitos foram capazes de acessar os exemplos relevantes, entendê-los e completar a tarefa de programação. Estes resultados forneceram evidências iniciais de que a correspondência estrutural é apropriada para obter exemplos relevantes e ajudar na atividade de reuso de um *framework*.

Este trabalho se mostrou importante para o trabalho apresentado nesta tese por mostrar que repositórios de exemplos é uma boa solução para ajudar na instanciação de *frameworks*, e discutir sobre o uso de exemplos, vantagens e desvantagens. O trabalho de Holmes e Murph também se mostrou importante por fornecer um conjunto inicial de heurísticas para determinar semelhança estrutural, que pode ser aplicado em um possível trabalho futuro. Pois, a abordagem desta tese apresenta exemplos de código com os *hot-spots* usados na instanciação de características, mas não foi implementada a escolha do melhor exemplo e uma técnica como esta apresentada em (HOLMES; MURPHY, 2005)

poderia ser aplicada neste propósito, na busca pelo exemplo que tem maior aderência com o que está sendo implementado pelo desenvolvedor.

Abordagens com repositórios de exemplos tem duas limitações: elas impõem uma grande sobrecarga para o desenvolvedor ao consultar o repositório, e construir o repositório exige exemplos cuidadosamente construídos ou código bem comentados. Esta é uma vantagem da abordagem proposta nesta tese, que não depende de um conhecimento aprofundado sobre o *framework* e exemplos, e não depende que o desenvolvedor conheça outras linguagens de consulta. O desenvolvedor precisa apenas saber executar a característica nos sistemas exemplos.

Outro trabalho baseado em exemplos é (BRUCH; SCHäFER; MEZINI, 2006), Bruch e colegas propoem o uso de técnicas de mineração de dados para extrair padrões de reutilização de instâncias de *frameworks* existentes. Com base nesses padrões, a abordagem sugere outras partes relevantes do *framework* para os usuários novatos em uma forma dependente do contexto. Os autores apresentam uma ferramenta protótipo chamada FrUiT, um *plug-in* do Eclipse que implementa esta abordagem e apresentam uma primeira avaliação por meio da mineração das partes do *framework* Eclipse. A FrUiT "aprende" como usar um *framework* analisando instâncias existentes. Esta abordagem combina o uso de técnicas de mineração de dados (MICHAIL, 2000) com uma apresentação dependente de contexto (HOLMES; MURPHY, 2005). Com base em instâncias de *frameworks* existentes, regras de reutilização que ocorrem com frequência são extraídos e armazenados numa base de dados. Usuários de *frameworks* podem então consultar automaticamente as regras que são relevantes no contexto da sua atividade atual.

Para encontrar as regras de reutilização que ocorrem com frequência, depois de criar um conjunto de propriedades de classe para cada classe nos exemplos de instanciação, a abordagem apresenta três fases: **1)** primeiro, extrair informações a partir dos exemplos de instanciação do *framework*; **2)** em segundo lugar, mineração de dados para extrair regras para reutilização com base nas informações sobre instanciação coletadas na primeira fase; **3)** em terceiro lugar, aplicar filtros sobre as regras encontradas para remover aquelas que não são relevantes para a compreensão do *framework*.

A partir de um ponto de partida, que são as funcionalidade do *framework* desejado, FrUiT apresenta elementos de programação relevantes para o usuário, regras que definem se o usuário deve criar subclasses que estendem classes do *framework*, implementação de interfaces do *framework*, sobrescrever métodos, chamadas de método e instancias, que representa uma chamada para um construtor de dentro da classe do *framework*. A Figura 38 apresenta uma tela da ferramenta FrUiT, onde será inciada a criação de uma nova *WizardDialog*. O item 1 na Figura 38 é a criação da *WizardDialog* no Eclipse. O item 2 na Figura 38 mostra uma analise de código-fonte com o FrUiT e as dicas de implementação mostrados na visão de sugestão. A visão lógica, item 3 na Figura 38, mostra os pré-requisitos da regra selecionada na visão sugestão. No item item 4 na Figura 38, os autores

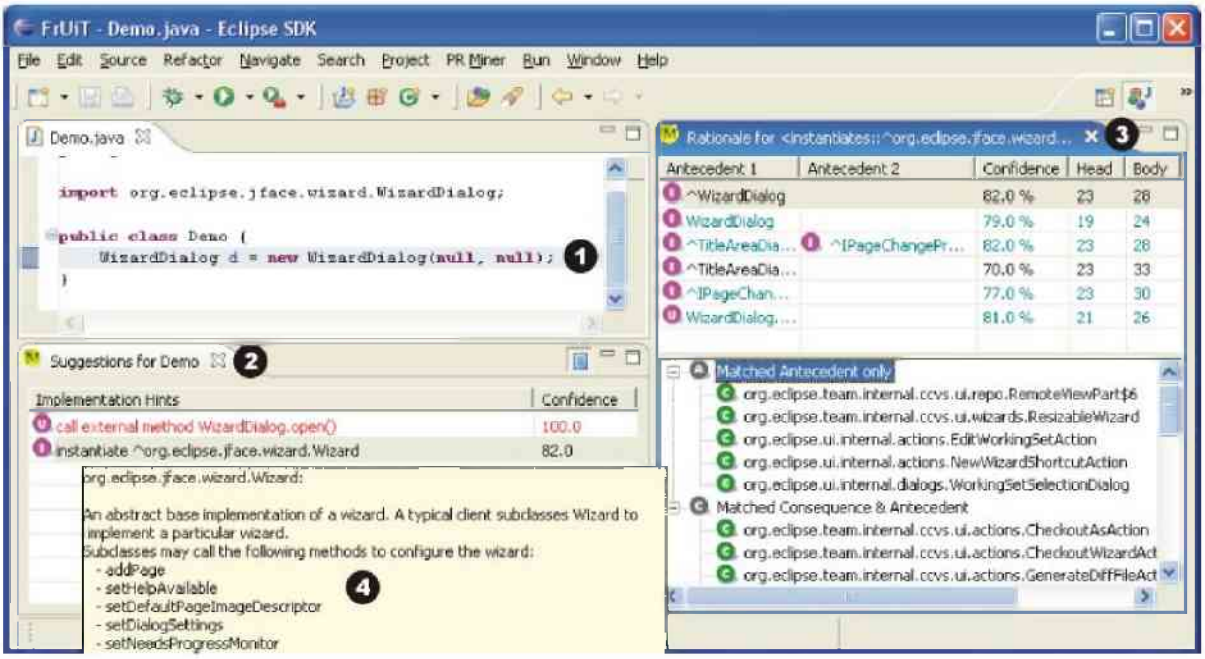


Figura 38 – Exemplo deu uso da ferramenta FrUiT (BRUCH; SCHäFER; MEZINI, 2006).

apontam o *mouse* sobre a sugestão para revelar o comentário JavaDoc correspondente.

FrUiT trata o problema de localizar as regras de interesse utilizando mineração de dados, uma técnica estática para obtenção dos dados. Nas avaliações, FrUiT apresenta os elementos do programa relevantes para o usuário e alivia os desenvolvedores de procurar regras de interesse. Em vez disso, a ferramenta usa o contexto atual e apresenta regras relevantes automaticamente. No entanto, em contraste com o trabalho apresentado nesta tese, FrUiT não fornece trechos de código ou um modelo, receita de implementação completo. Além disso, a abordagem FrUiT é útil quando o desenvolvedor conhece pelo menos os elementos de API de interesse; este é menos útil se o desenvolvedor tem apenas uma ideia de alto nível do conceito de que deve ser aplicado.

No trabalho (TSANTALIS et al., 2006), que será apresentado a seguir, Tsantalis e colegas propõem uma metodologia para detecção de padrões de projeto, baseado na similaridade entre vértices de gráficos. Este trabalho foi apresentado anteriormente por ser utilizado na abordagem apresentada nesta tese. Devido à natureza do algoritmo gráfico subjacente, esta abordagem tem a capacidade de reconhecer também padrões que são modificados a partir da sua representação padrão. Além disso, a abordagem explora o fato que os padrões de projeto podem residir em um ou mais hierarquias de herança, reduzindo o tamanho dos gráficos para que o algoritmo seja aplicável.

Na metodologia proposta Tsantalis e colegas, tanto o sistema em estudo, bem como o padrão de projeto a ser detectado é descrito em termos de gráficos. A abordagem emprega um conjunto de matrizes que representam todos os aspectos importantes da estrutura estática. A idéia fundamental é que o diagrama de classes é, essencialmente, um grafo

direcionado que pode ser perfeitamente mapeados em uma matriz. As duas principais vantagens desta abordagem são que as matrizes podem ser facilmente manipulados, e que este tipo de representação é intuitivamente atraente para os engenheiros e cientistas da computação. Para a detecção de padrões empregam um algoritmo de similaridade entre gráfico (BLONDEL et al., 2004), que utiliza como entrada tanto os gráficos do sistema e dos padrões, então calcula pontuações de semelhança entre os seus vértices. A principal vantagem desta abordagem é a capacidade de detectar não apenas os padrões na sua forma base (esta é uma abordagem normalmente encontrada na literatura), mas também versões modificadas dos mesmos (dado que a modificação é limitada a um padrão característico). Este é um pré-requisito importante uma vez que qualquer padrão de projeto pode ser implementado com variações.

Um problema que requer tratamento cuidadoso é que a convergência do algoritmo de semelhança depende do tamanho do sistema de gráfico. Para lidar com este problema, a abordagem de (TSANTALIS et al., 2006) explora o fato de que cada padrão de projeto reside em uma ou mais hierarquias de herança, uma vez que a maioria dos padrões envolvem, pelo menos, uma interface abstrata e seus descendentes. Por conseguinte, o sistema é particionado em *clusters* de hierarquias (pares e hierarquias que se comunicam), de modo que o algoritmo de similaridade é aplicada a subsistemas menores em vez de todo o sistema.

Os autores desenvolveram um programa Java que automatiza a metodologia citada e gera uma lista das instâncias de padrões detectados. O programa emprega um *framework* manipulação de *bytecode* Java que fornece informações detalhadas sobre a estrutura estática do sistema. As matrizes que representam o sistema em estudo são construídos de acordo com essa informação. A metodologia proposta automatiza completamente o processo de detecção do padrão de extração dos casos reais. As informações que os autores escolheram para a representação inclui associações, generalizações, classes abstratas, criações de objetos, abstrato invocações de método, etc.

Os autores avaliaram a metodologia proposta nos *frameworks* JHotDraw e JUnit [4], e com a ferramenta JRefactory [5], que são projetos de código aberto que amplamente e sistematicamente empregam padrões de projeto. Os resultados foram validados comparando os resultados com as documentação internas e externas desses sistemas. Para os padrões de projeto examinados, o número de falsos negativos foi limitado, enquanto os falsos positivos não foram encontrados.

Aprincipal contribuição da abordagem é o uso de um algoritmo de similaridade, que tem a vantagem inerente de detectar mudanças nos padrões, que podem ocorrer. A abordagem e ferramental de Tsantalis e colegas foram escolhidos para compor a abordagem apresentada nesta proposta, na etapa de obtenção dos padrões de projeto do *framework*.

[4] http://www.junit.org, 2006
[5] http://jrefactory.sourceforge.net/

A escolha foi motivada pela sua avaliação em três projetos open-source que demonstrou a precisão e a eficiência do método proposto. Além do mais, os autores realizaram experimentos com o *framework* JHotDraw, versão 5.3, que também foi utilizado nas avaliações da abordagem apresentada nesta tese, o que permitiu verificar a confiabilidade e utilidade dos dados gerados pela abordagem de Tsantalis e colegas em trabalhos preliminares.

Por último, segue o trabalho de HEYDARNOORY (2012) que introduz a noção de *templates* de implementação de conceitos e uma ferramenta chamada FUDA (*Framework API Understanding through Dynamic Analysis*), uma abordagem para a extração automática de tais *templates* a partir de rastros de execução de aplicações exemplo. Conceitos são unidade genéricas de funcionalidades. Um *template* de implementação de um conceito é como um pseudo-código em Java demonstrando as etapas de implementação, que são necessárias para criar uma instância de um determinado conceito.

FUDA obtém informações contidas em aplicativos que utilizam o *framework* para entender a implementação de um conceito desejado. Para este fim, utiliza apenas a análise dinâmica para determinar as partes do código das aplicações exemplo que são relevantes para a implementação desse conceito. Vale ressaltar que a técnica de *slicing* dos rastros é completamente diferente das abordagens tradicionais de *slincing.* O corte dos rastro das APIs trabalha com rastros de interações entre as aplicações e o *framework*, enquanto as técnicas tradicionais de corte trabalham com rastros de instruções do programa. Além disso, no *slicing* tradicional a dependência entre os eventos é definido em termos de dependências de dados e de controle. No *slicing* usado por Heydarnoory, a dependência entre eventos é definida em termos de uso comum dos objetos como alvos, parâmetros, ou valores de retorno das chamadas.

Um *template* de implementação especifica quais pacotes de *framework* importar, quais classes do *framework* estender, interfaces para implementar, e as operações que devem ser chamadas. Este *template* pode ser usado como um resumo conciso das etapas necessárias para implementar os conceito [6] e um ponto de partida para investigar as implementações concretas dos conceitos nas aplicações exemplo. A Figura 39 apresenta um exemplo de *template* para o *menu* de contexto do *framework* Eclipse.

Este trabalho apresentou uma ferramenta protótipo que foi testada em doze conceitos de *frameworks* amplamente utilizados. A amostra incluiu conceitos simples e complexos. Seis conceitos correspondem a questões encontradas em fóruns de desenvolvimento. Os *templates* gerados, exceto um, obtiveram precisão de pelo menos 78% e cobertura (*recall*) de pelo menos 89%. Além disso, mais da metade dos *templates* tinha precisão de 90% ou melhor e cobertura de 100%. Em geral, os falsos positivos foram mais frequentes do que os falsos negativos, particularmente quando apenas um rastro foi usado. Quando apenas um único sistema exemplo é usado, os tamanhos dos *templates* são maiores do que quando dois ou três sistemas de exemplo são usados porque FUDA usa a interseção

[6] do inglês concern

```
1  import org.eclipse.jface.action.Separator;
2  import org.eclipse.jface.viewers.Viewer;
3  import org.eclipse.jface.action.Action;
4  import org.eclipse.jface.action.MenuManager;
5  import org.eclipse.swt.widgets.Menu;
6  import org.eclipse.jface.resource.ImageDescriptor;
7  import org.eclipse.jface.action.IMenuListener;
8  import org.eclipse.swt.widgets.Control;

9  public class AppMenuListener implements IMenuListener {                    (l. 204)→
10   public void menuAboutToShow(menuManager) {                                (l. 205)→
11     Separator separator = new Separator(String)|||(); //REPEAT      (l. 215 l. 217)→
12     menuManager.add(separator)|||(appAction); //REPEAT      (l. 213 l. 215 l. 217)→
13   }
14 }

15 public class AppAction extends Action {                                (l. 220 l. 225)→
16 }

17 public class SomeClass {
18   public void someMethod() {
19     Viewer viewer = ...;
20     Control control = viewer.getControl(); //MAY REPEAT              (l. 208 l. 209)→
21     AppAction appAction = new AppAction(); //MAY REPEAT             (l. 220 l. 225)→
22     appAction.setText(String); //MAY REPEAT                         (l. 223 l. 228)→
23     appAction.setToolTipText(String); //MAY REPEAT                  (l. 224 l. 229)→
24     MenuManager menuManager =
25       new MenuManager(String)|||(String,String)|||();                       (l. 202)→
26     menuManager.setRemoveAllWhenShown(boolean);                             (l. 203)→
27     AppMenuListener appMenuListener = new AppMenuListener();                (l. 204)→
28     menuManager.addMenuListener(appMenuListener);                           (l. 204)→
29     Menu menu = menuManager.createContextMenu(control);                     (l. 208)→
30   }
31 }
```

Figura 39 – Exemplo *template* da ferramenta FUDA (HEYDARNOORI et al., 2012)

entre os rastreamentos. No entanto, eles são poluídos com falsos positivos, de modo que a precisão para a maioria das características é de cerca de 50% ou menos. A cobertura é alto, na faixa de 92-100%, como nenhum evento é removido dos vestígios devido à interseção.

Em nossa abordagem a cobertura foi de 100% para as classes e métodos do *framework* JHotDraw com nenhum falso positivo e nenhum falso negativo no Livro de Receitas gerado. No caso do livro de Receitas do Jface, a cobertura foi de 100% das classes e 95% dos métodos apresentando 0,5% de falsos negativos, devido à um método que não foi obtido no rastro e ocorreu em várias características avaliadas. Sobre os falsos positivos, ocorreu 0% de elementos para os dois *frameworks*. Sendo a maioria dos resultados obtidos a partir de um único rastro de um único sistema.

Na definição da abordagem apresentada nesta tese, percebeu-se que o as informações estáticas presentes no RSF sobre as relações entre classes da aplicação (SPECIFIC) e *hot-spots* (*FRAMEWORK*) auxiliaram na diminuição do número de falsos positivos. Na primeira versão da abordagem, esta informação não foi utilizada e verificamos os resultados para o *framework* JHotDraw, onde falsos positivos foram constatados. Para a

característica "Criar uma nova figura" deste *framework*, o número de classes *hot-spots* na receita foi 24 para a primeira versão. Este número caiu para 9 (15 falsos positivos, 62% de falsos positivos) para a última versão utilizando os dados estáticos e com um único rastro. No entanto, a cobertura sempre foi acima de 95%. O uso da análise estática junto à análise dinâmica foi importante para melhorar a precisão, diminuindo o número de falsos positivos para zero em nossos testes. E, comparando aos resultados de Heydanoory, tivemos uma precisão aparentemente melhor com o uso de um único rasto. Não realizamos uma análise comparativa sistemática com esta abordagem, envolvendo as mesmas características e *frameworks*. Porém, para o *Framework* JHotDraw, duas características diferentes das nossas foram testadas e obtiveram precisão de 71% e 71% com cobertura para a primeira (2 falsos positivos e 2 falsos negativos), e 85% de precisão e 89% de cobertura para a segunda (3 falsos positivos e 2 falsos negativos). Para o *framework* Eclipse, duas características também avaliadas por nós, criar tabela e criar árvores, apresentaram respectivamente 100% de precisão e 96% de cobertura (0 falsos positivos e 2 falsos negativos) e 100% de precisão e 98% de cobertura (0 falsos positivos e 1 falsos negativos). Porém, Heydanoory utiliza mais de um sistema para gerar os *templates* que apresentam uma estrutura diferente da nossa como se pode ver na Figura 39. Os tipos de informações apresentadas nos *templates* da abordagem FUDA também são apresentados nas receitas da abordagem apresentada nesta tese, o que confirma que a escolha das informações que compõem as receitas foi adequado. Porém, as receitas apresentam informações relativas à padrões de projeto utilizados pelo *framework*,comentários e exemplos de uso extraídos do código, que não são apresentados diretamente no *template* da abordagem FUDA.

Em um experimento com doze usuários, não houve diferença estatisticamente significativa entre o uso dos *templates* verso o uso dos documentos, em termos de tempo de implementação e o número de *bugs* introduzidos que puderam ser detectados. O autor afirma que os *templates* FUDA são comparáveis a boa documentação estrutura que fornecem descrições completas. Em particular, para as amostras estudadas, a escolha dos *templates* versus a documentação teve muito menos impacto sobre o tempo de desenvolvimento do que a complexidade do conceito. O autor conclui em seu trabalho, que os *templates* FUDA podem servir como um substituto para a documentação quando não existe documentação disponível. No entanto, este estudo com sujeitos também sugeriu que os *templates* devem ser usados em conjunto com as aplicações exemplo de onde os dados foram extraídos de forma a obter mais informações sobre a implementação dos conceitos e para detectar falsos positivos e negativos nos modelos.

Este capitulo apresentou alguns dos principais trabalhos relacionados e suas principais contribuições para a abordagem definida para esta tese.

Capítulo 7

Conclusão

Frameworks são amplamente utilizados e são uma forma de reúso de software efetiva. Porém, eles apresentam uma grande curva de aprendizado devido a sua complexidade, grande quantidade classes abstratas (*hot-spots*) a serem entendidas de maneira inter-relacionada e um *design* de projeto pré-estabelecido que deve ser compreendidos para tornar o reúso efetivo. Uma boa documentação poderia ajudar neste problema, mas estas frequentemente não existem ou não são apropriadas (HEYDARNOORI et al., 2012). Alguns *frameworks* apresentam pouca documentação que não seja o código-fonte e um conjunto de exemplos (JOHNSON, 1997) (HEYDARNOORI et al., 2012).

JOHNSON (1992) ainda afirma que muitas vezes, a melhor maneira de aprender a gama de aplicabilidade de um *framework* é por meio de exemplo. Os resultados de SHULL et al. (2000) já haviam sugerido que os exemplos são uma estratégia de aprendizagem eficaz, especialmente para aqueles começando a aprender um *framework*. Porém, analisar até mesmo um pequeno conjunto de exemplos de código manualmente é um processo trabalhoso e propenso a erros (COTTRELL et al., 2009), dado o grande tamanho dos *frameworks* típicos, a complexidade de suas interfaces, e ao grande número de instanciações possíveis. Então, mesmo quando se pensa no código existente como uma possível documentação, o processo de aprendizado pode ser custoso. Este cenário motiva a criação de uma documentação que organize os interesses de instanciação e apresente exemplos associados aos mesmos.

A ideia central da abordagem proposta nesta tese é que as instruções necessárias para criar uma instância de um *framework* para características desejadas podem ser obtidos por engenharia reversa realizada sobre sistemas que utilizam o respectivo *framework*. Esta proposta é apoiada pelo crescente uso de repositórios de software para fornecer exemplos de código e sistemas.

Nesta tese foi proposto o uso de livro de receitas como documento para instanciação de *frameworks*, onde as receitas são uma documentação estruturada por característica desejadas, que apresentam uma lista de tarefas a serem executadas com instruções importantes sobre os *hot-spots* e exemplos de uso que visam auxiliar na compreensão do que deve ser

feito para a instanciação. Para definir a estrutura deste livro de receitas, consideramos quatro categorias de problemas que ocorrem durante a reutilização de *framework*, identificadas em (KIRK; ROPER; WOOD, 2005): **i)** o mapeamento (localização do código-fonte de interesse) da solução para o *framework*; **ii)** compreender as interações dentro do *framework*; **iii)** a compreensão das características que estão disponíveis no *framework*; **iv)** e manter a consistência arquitetônica do *framework* durante uma modificação. Utilizou-se análise dinâmica para localizar o código fonte ligado a instanciação das características de interesse nos exemplos, e assim, criar um Livro de Receitas para as características que podem ser instanciadas usando o *framework*, e apresenta os elementos de código ligados à esta instanciação. Utilizou-se análise estática para trazer informações sobre os elementos de código e suas interações dentro do *framework*, incluindo exemplos de código para servirem de exemplos de instanciação. Para manter a consistência arquitetônica do *framework* incluímos informações sobre padrões de projeto.

A localização do código que implementa a característica desejada no código-fonte de aplicações exemplo pode ser um desafio, devido ao código desejado estar muitas vezes espalhado e/ou entrelaçado com código de execução de outras características, conforme apresentado e exemplificado na introdução deste trabalho (MAIA; LAFETá, 2013). Pelo melhor do nosso conhecimento, a abordagem aqui apresentada e a de HEYDARNOORY (2012) são as únicas que utilizam análise dinâmica para localizar as características e apresentar uma abordagem que auxilie neste propósito. Porém, abordagens dinâmicas apresentam problemas com falsos negativos e falsos positivos e acreditamos que o uso, em conjunto, de abordagem estática pode minimizar este problema e trazer informações úteis para os Livros de receitas.

O diferencial do trabalho proposto nesta tese, pelo melhor do nosso conhecimento, esta é a única abordagem híbrida, estática e dinâmica, para obter informações que auxiliem na instanciação de *frameworks*. O conjunto de informações que compõem as receitas também é um diferencial, pois é o único que apresenta tarefas, informações, padrões de projeto e exemplos de instanciação em uma única documentação direcionada por características de interesse.

A taxa de cobertura dos livros de receitas para a versão final da abordagem foi avaliada para os *frameworks* JHotDraw e JFace. Para o JHotDraw, 100% das classes e métodos *hot-spot* esperados foram apresentados nas receitas. Para o JFace, 100% das classes *hot-spots* e mais de 95% dos métodos *hot-spots* esperados foram apresentados. O Livro de Receitas do JFace apresentou 0,5% de Falsos Positivos. Estas taxas foram obtidas comparando as informações apresentadas nos Livros de Receitas com os exemplos de implementação das características em sistemas que utilizam os *frameworks*.

Esta tese mostrou uma abordagem semi-automatizada capaz de gerar livros de receitas, pelo menos tão bons quanto as documentações tradicionais durante o uso em atividades reais de instanciação, em termos do tempo de execução, taxa de acerto e satisfação ao

utilizar o documento. O tempo de execução não apresentou significante diferença em nenhum experimento. A taxa de acerto foi significantemente melhor para o grupo Receitas no experimento com alunos Instituição 1. Para os demais experimentos apresentou um resultado sem significante diferença entre os grupos. A percepção de satisfação de uso do documento foi maior para todos os experimentos. Este resultado pode ter sido influenciado pela percepção utilidade das informações, facilidade de localização das informações, utilidade do mecanismo de navegação e clareza das informações, que foram melhores ou iguais ao uso dos documentos tradicionais.

Estes resultados permitem a adoção desta abordagem de construção de documentação para as situações onde falta documentação ou existe documentação desatualizada do *framework* que se deseja utilizar. Para os desenvolvedores de *frameworks*, esta abordagem pode auxiliar no desenvolvimento da documentação, uma vez que os *frameworks* são testados com a implementação de sistemas exemplo, a partir dos quais os livros de receitas poderiam ser gerados.

Foram realizados 3 experimentos envolvendo ao todo 44 sujeitos humanos, com 88 execuções de atividades reais de instanciação de *frameworks*, onde o uso de livros de receitas foram comparados ao uso de documentações tradicionais dos *frameworks* na realização de atividades reais de instanciação. Experimentos com sujeitos são menos encontrados na literatura devido à dificuldade em realiza-los (CORNELISSEN; ZAIDMAN; DEURSEN, 2011),(MAIA; LAFETá, 2013). A aplicação de atividades reais de desenvolvimento é mais raro ainda, a maioria dos experimentos com sujeitos apresentam questões de compreensão, Como se pode ver na Tabela 6.

Os relatos dos sujeitos indicaram a importância dos exemplos de código para execução das atividades. Os sujeitos profissionais apresentaram uma preferência por utilizarem os exemplos de código presentes nos sistemas exemplo ao invés de utilizar as documentações. É importante ressaltar que os livros de receitas ajudaram na localização dos *hot-spots* e exemplos de códigos para estes profissionais,

Portanto, os seus objetivos propostos foram cumpridos ao apresentar uma abordagem semi-automatizada e uma nova ferramenta chamada CookFrame capaz de gerar livros de receitas como alternativa viável de documentação para guiar o desenvolvedor durante o processo de instanciação. Quando comparados às documentações tradicionais dos *frameworks*, para os experimentos realizados, os livros de receita apresentaram resultados melhores ou iguais ao uso destes.

7.1 Trabalhos Futuros

Neste capítulo iremos apresentar dois possíveis trabalhos futuros relacionados à abordagem proposta por esta tese.

Pretende-se aplicar a abordagem sobre o *framework* Android para obter receitas es-

pecíficas e compará-las com documentações sobre as características de interesse, dada a importância econômica deste *framework*. Até o momento, as ferramentas utilizada para gerar os dados estáticos (Java2RSF) e dinâmicos (TraceExtractor) não funcionaram bem para os testes sobre as aplicações que utilizam o *framework* Android, devido estas aplicações não utilizarem apenas a linguagem Java. Mas, em uma avaliação preliminar, percebe-se que a linguagem de mineração de informações BOA [1] pode ser utilizada como uma alternativa para obter as informações estáticas necessárias pra a abordagem. A mesma tecnologia AspectJ utilizada para criar o TraceExtractor (SOBREIRA; MAIA, 2008), pode também ser adaptada para o *framework* Android, viabilizando este trabalho futuro.

ROCHA e MAIA (2016) criaram uma abordagem para gerar tutoriais para APIs a partir do Stack Overflow e organizá-los conforme a complexidade de entendimento. As metodologias foram avaliadas por meio de tutoriais gerados para a API Swing. O resultado geral da avaliação foi positivo, mostrando a viabilidade para uso de tutoriais gerados automaticamente a partir do Stack Overflow. Um possível trabalho futuro é tentar unir as informações obtidas automaticamente pela abordagem de Rocha e Maia com as informações obtidas semi-automaticamente pela abordagem proposta nesta tese para auxiliar no uso de APIs. Devido à percepção de que os desenvolvedores preferem ferramentas que tragam informações mais diretas ao seu interesse do que documentações que devem ser lidas para serem consultadas, a proposta seria criar uma ferramenta de recomendação que quando utilizada traga as informações conforme o entradas sugeridas pelo o usuário.

[1] http://web.cs.ucla.edu/ shyoo1st/boa/

Referências

ALEXANDER, S. I. C.; SILVERSTEIN WITH MAX JACOBSON, I. F.-K. M.; ANGEL, S. **A Pattern Language**. New York: Press, 1977.

ANTONIOL, G.; GUEHENEUC, Y.-G. Feature identification: An epidemiological metaphor. **IEEE Transactions on Software Engineering**, v. 32, n. 9, p. 627–641, Sept 2006.

BALL, T. The concept of dynamic analysis. **Proceedings of the ACM SIGSOFT International Symposium on Foundations of Software Engineering**, ACM, New York, NY, USA, v. 24, n. 6, p. 216–234, out. 1999. ISSN 0163-5948.

BATORY, D. A tutorial on feature oriented programming and the ahead tool suite. In: **Proceedings of the 2005 International Conference on Generative and Transformational Techniques in Software Engineering (GTTSE'05)**. Berlin, Heidelberg: Springer-Verlag, 2006. (GTTSE'05), p. 3–35.

BLOCH, J. **Effective Java Programming Language Guide**. Mountain View, CA, USA: Sun Microsystems, Inc., 2001.

BLONDEL, V. D. et al. A measure of similarity between graph vertices: Applications to synonym extraction and web searching. **SIAM Rev.**, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, v. 46, n. 4, p. 647–666, abr. 2004.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **Unified Modeling Language User Guide, The (2Nd Edition) Addison-Wesley Object Technology Series**. [S.l.]: Addison-Wesley Professional, 2005.

BRIAND, L. C.; LABICHE, Y.; LEDUC, J. Toward the reverse engineering of uml sequence diagrams for distributed java software. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 32, n. 9, p. 642–663, set. 2006.

BRUCH, M.; SCHäFER, T.; MEZINI, M. Fruit: Ide support for framework understanding. In: **Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (OOPSLA'2006)**. New York, NY, USA: ACM, 2006. (eclipse '06), p. 55–59.

CHIKOFSKY, E.; CROSS J.H., I. Reverse engineering and design recovery: a taxonomy. **IEEE Software**, v. 7, n. 1, p. 13–17, Jan 1990.

COOK, J. E.; DU, Z. Discovering thread interactions in a concurrent system. In: **Proceedings of the 9th Working Conference on Reverse Engineering, 2002.** [S.l.: s.n.], 2002. p. 255–264.

CORNELISSEN, B.; ZAIDMAN, A.; DEURSEN, A. van. A controlled experiment for program comprehension through trace visualization. **IEEE Transactions on Software Engineering**, v. 37, n. 3, p. 341–355, May 2011.

CORNELISSEN, B. et al. A systematic survey of program comprehension through dynamic analysis. **IEEE Transactions on Software Engineering**, v. 35, n. 5, p. 684–702, Sept 2009.

COTTRELL, R. et al. Compare and contrast: Visual exploration of source code examples. In: **Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'2009).** [S.l.: s.n.], 2009. p. 29–32.

DAGENAIS, B.; OSSHER, H. Automatically locating framework extension examples. In: **Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering**. New York, NY, USA: ACM, 2008. (SIGSOFT '08/FSE-16), p. 203–213.

DYER, R. et al. Boa: Ultra-large-scale software repository and source-code mining. **ACM Transactions on Software Engineering and Methodology.**, ACM, New York, NY, USA, v. 25, n. 1, p. 7:1–7:34, dez. 2015. ISSN 1049-331X.

EISENBARTH, T.; KOSCHKE, R.; SIMON, D. Locating features in source code. **IEEE Transactions on Software Engineering**, v. 29, n. 3, p. 210–224, March 2003.

FAIRBANKS, G.; GARLAN, D.; SCHERLIS, W. Design fragments make using frameworks easier. In: **Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)**. New York, NY, USA: ACM, 2006. (OOPSLA '06), p. 75–88.

FAYAD, M. E.; SCHMIDT, D. C.; JOHNSON, R. E. **Building Application Frameworks: Object-oriented Foundations of Framework Design**. New York, NY, USA: John Wiley & Sons, Inc., 1999.

FEIGENSPAN, J. et al. Measuring programming experience. In: **Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC'2012)**. [S.l.: s.n.], 2012. p. 73–82.

GAMMA, E.; BECK, K. **Contributing to Eclipse: Principles, Patterns, and Plugins**. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 2003.

GAMMA, E. et al. **Design Patterns: Elements of Reusable Object-oriented Software**. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

GEORGAKOPOULOS, D.; HORNICK, M.; SHETH, A. An overview of workflow management: From process modeling to workflow automation infrastructure. In: **Distributed and Parallel Databases**. [S.l.: s.n.], 1995. p. 119–153.

GREEVY, O.; DUCASSE, S.; GIRBA, T. Analyzing feature traces to incorporate the semantics of change in software evolution analysis. In: **Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM'2005).** [S.l.: s.n.], 2005. p. 347–356.

HAMOU-LHADJ, A.; LETHBRIDGE, T. C. A survey of trace exploration tools and techniques. In: **Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (CASCON'2004)**. [S.l.]: IBM Press, 2004. (CASCON '04), p. 42–55.

HARDER, J.; TIARKS, R. A controlled experiment on software clones. In: **Proceedings of the 20th IEEE International Conference on Program Comprehension (ICPC'2012).** [S.l.: s.n.], 2012. p. 219–228.

HENNINGER, S. Retrieving software objects in an example-based programming environment. In: **Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '91)**. New York, NY, USA: ACM, 1991. (SIGIR '91), p. 251–260.

HERMANS, F.; AIVALOGLOU, E. Do code smells hamper novice programming? a controlled experiment on scratch programs. In: **Proceedings of the IEEE 24th International Conference on Program Comprehension (ICPC'2016)**. [S.l.: s.n.], 2016. p. 1–10.

HEUZEROTH, D.; HOLL, T.; LÖWE, W. Combining static and dynamic analyses to detect interaction patterns. In: **Proceedings of the 16th International Conference on Integrated Design and Process Technology (IDPT'2002)**. [S.l.: s.n.], 2002. p. 2.

HEYDARNOORI, A. et al. Two studies of framework-usage templates extracted from dynamic traces. **IEEE Transactions on Software Engineering**, v. 38, n. 6, p. 1464–1487, Nov 2012.

HOLMES, R.; MURPHY, G. Using structural context to recommend source code examples. In: **Proceedings of the 27th International Conference on Software Engineering (ICSE'2005).** [S.l.: s.n.], 2005. p. 117–125.

HOU, D.; WONG, K.; HOOVER, H. What can programmer questions tell us about frameworks? In: **Proceedings of the 13th International Workshop on Program Comprehension (IWPC'2005).** [S.l.: s.n.], 2005. p. 87–96.

JAVED, M. A.; ZDUN, U. The supportive effect of traceability links in architecture-level software understanding: Two controlled experiments. In: **Proceedings of the IEEE/IFIP Conference on Software Architecture, 2014.** [S.l.: s.n.], 2014. p. 215–224.

JIANG, J. et al. Constructing usage scenarios for api redocumentation. In: **Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC'2007).** [S.l.: s.n.], 2007. p. 259–264.

JOHNSON, R. E. Documenting frameworks using patterns. In: **Proceedings of the Conference on Object-oriented Programming Systems, Languages, and**

Applications (OOPSLA'1992). New York, NY, USA: ACM, 1992. (OOPSLA '92), p. 63–76.

_____. Components, frameworks, patterns. In: **Proceedings of the 1997 Symposium on Software Reusability (SSR'1997)**. New York, NY, USA: ACM, 1997. (SSR '97), p. 10–17.

KIRK, D.; ROPER, M.; WOOD, M. Identifying and addressing problems in framework reuse. In: **Proceedings of the 13th International Workshop on Program Comprehension (IWPC'2005).** [S.l.: s.n.], 2005. p. 77–86.

_____. Identifying and addressing problems in object-oriented framework reuse. **Empirical Software Engineering**, Kluwer Academic Publishers, Hingham, MA, USA, v. 12, n. 3, p. 243–274, jun 2007.

KOSKINEN, J.; KETTUNEN, M.; SYSTA, T. Profile-based approach to support comprehension of software behavior. In: **Proceedings of the IEEE 14th International Conference on Program Comprehension (ICPC'2006)**. [S.l.: s.n.], 2006. p. 212–224.

KOTHARI, J. et al. Reducing program comprehension effort in evolving software by recognizing feature implementation convergence. In: **Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC'2007).** [S.l.: s.n.], 2007. p. 17–26.

KRASNER, G. E.; POPE, S. T. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. **J. Object Oriented Program.**, SIGS Publications, Denville, NJ, USA, v. 1, n. 3, p. 26–49, aug 1988.

LUKOIT, K. et al. TraceGraph: Immediate visual location of software features. In: **Proceedings of the 16nd International Conference on Software Maintenance (ICSM'2000)**. [S.l.]: IEEE C.S., 2000. p. 33–39.

MAIA, M. d. A.; LAFETá, R. F. On the impact of trace-based feature location in the performance of software maintainers. **Journal of Systems and Software, 2013**, v. 86, n. 4, p. 1023 – 1037, 2013.

MAR, L. W.; WU, Y.-C.; JIAU, H. Recommending proper api code examples for documentation purpose. In: **Software Engineering Conference (APSEC), 2011 18th Asia Pacific**. [S.l.: s.n.], 2011. p. 331–338.

MARKIEWICZ, M. E.; LUCENA, C. J. P. de. Object oriented framework development. **Crossroads**, ACM, New York, NY, USA, v. 7, n. 4, p. 3–9, jul 2001.

MAYRHAUSER, A. von; LANG, S. A coding scheme to support systematic analysis of software comprehension. **IEEE Transactions on Software Engineering**, v. 25, n. 4, p. 526–540, Jul 1999.

MICHAIL, A. Data mining library reuse patterns using generalized association rules. In: **Proceedings of the International Conference on Software Engineering, 2000.** [S.l.: s.n.], 2000. p. 167–176.

NEAL, L. R. A system for example-based programming. **SIGCHI Bull.**, ACM, New York, NY, USA, v. 20, n. SI, p. 63–68, mar. 1989. ISSN 0736-6906.

NYKAZA, J. et al. What programmers really want: results of a needs assessment for sdk documentation. In: **SIGDOC**. [S.l.: s.n.], 2002.

OLIVEIRA, T. et al. Software process representation and analysis for framework instantiation. **IEEE Transactions on Software Engineering**, v. 30, n. 3, p. 145–159, 2004.

OLIVEIRA, T. C.; ALENCAR, P.; COWAN, D. Reusetool-an extensible tool support for object-oriented framework reuse. **Journal of Systems and Software, 2011.**, Elsevier Science Inc., New York, NY, USA, v. 84, n. 12, p. 2234–2252, dec 2011.

ORTIGOSA, A.; CAMPO, M. Smartbooks: a step beyond active-cookbooks to aid in framework instantiation. In: **Proceedings of Technology of Object-Oriented Languages and Systems, 1999.** [S.l.: s.n.], 1999. p. 131–140.

PAUW, W. D. et al. Visualizing the execution of java programs. In: **International Seminar of Revised Lectures on Software Visualization, 2002.** London, UK, UK: Springer-Verlag, 2002. p. 151–162.

PENTA, M. D.; STIREWALT, R. E. K.; KRAEMER, E. Designing your next empirical study on program comprehension. In: **Proceedings of the 15th IEEE International Conference on Program Comprehension (ICPC'2007)**. [S.l.: s.n.], 2007. p. 281–285.

POSHYVANYK, D. et al. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. **IEEE Transactions on Software Engineering, 2007.**, v. 33, n. 6, p. 420–432, June 2007.

PREE, W. **Design Patterns for Object Oriented Software Development**. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

QUANTE, J. Do dynamic object process graphs support program understanding? – a controlled experiment. In: **Proceedings of the 16th International Conference on Program Comprehension (ICPC'2008)**. [S.l.]: IEEE C.S., 2008. p. 73–82.

REISS, S.; RENIERIS, M. Encoding program executions. In: **Proceedings of the 23rd IEEE International Conference on Software Engineering (ICSE'2001).** [S.l.: s.n.], 2001. p. 221–230.

ROBILLARD, M. P. What makes apis hard to learn? answers from developers. **IEEE Software**, v. 26, n. 6, p. 27–34, Nov 2009. ISSN 0740-7459.

ROBILLARD, M. P.; WEIGAND-WARR, F. Concernmapper: simple view-based separation of scattered concerns. In: **Proceedings of the OOPSLA workshop on Eclipse technology eXchange (Eclipse'2005)**. New York, NY, USA: ACM, 2005. p. 65–69.

ROCHA, A. M.; MAIA, M. A. Automated api documentation with tutorials generated from stack overflow. In: **Proceedings of the 30th Brazilian Symposium on Software Engineering (SBES'2016)**. New York, NY, USA: ACM, 2016. (SBES '16), p. 33–42.

ROMANO, S. et al. Are unreachable methods harmful? results from a controlled experiment. In: **Proceedings of the 24th IEEE International Conference on Program Comprehension (ICPC'2016)**. [S.l.: s.n.], 2016. p. 1–10.

RONG, G. et al. The impacts of supporting materials on code reading: A controlled experiment. In: **Proceedings of the Asia-Pacific Software Engineering Conference (APSEC'2015)**. [S.l.: s.n.], 2015. p. 88–95.

SHULL, F.; LANUBILE, F.; BASILI, V. Investigating reading techniques for object-oriented framework learning. **IEEE Transactions on Software Engineering**, v. 26, n. 11, p. 1101–1118, Nov 2000.

SOBREIRA, V.; MAIA, M. d. A. Analyzing feature scattering with visual information of execution traces. **INFOCOMP(UFLA)**, UFLA, Lavras, MG, Brazil, p. 21–30, 2008.

SOMMERVILLE, I. **Software Engineering**. [S.l.]: Pearson, 2011. (International Computer Science Series).

STEIDL, D.; HUMMEL, B.; JUERGENS, E. Quality analysis of source code comments. In: **Proceedings of the 21st IEEE International Conference on Program Comprehension (ICPC'2013)**. [S.l.: s.n.], 2013. p. 83–92.

SYSTA, T. On the relationships between static and dynamic models in reverse engineering java software. In: **Proceedings of the 6th Working Conference on Reverse Engineering, 1999.** [S.l.: s.n.], 1999. p. 304–313.

SYSTä, T.; KOSKIMIES, K.; MüLLER, H. Shimba-an environment for reverse engineering java software systems. **Software Practice and Experience**, John Wiley & Sons, Inc., New York, NY, USA, v. 31, n. 4, p. 371–394, abr. 2001. ISSN 0038-0644.

TSANTALIS, N. et al. Design pattern detection using similarity scoring. **IEEE Transactions on Software Engineering**, v. 32, n. 11, p. 896–909, Nov 2006.

VLISSIDES, J. **Protection, part i: The hollywood principle. C++ Report**. [S.l.: s.n.], 1996.

WEISER, M. Programmers use slices when debugging. **Commun. ACM**, ACM, New York, NY, USA, v. 25, n. 7, p. 446–452, jul. 1982. ISSN 0001-0782.

WETTEL, R.; LANZA, M.; ROBBES, R. Software systems as cities: a controlled experiment. In: **Proceedings of the IEEE 33rd International Conference on Software Engineering (ICSE'2011)**. [S.l.: s.n.], 2011. p. 551–560.

WILDE, N.; SCULLY, M. C. Software reconnaissance: Mapping program features to code. **Journal of Software Maintenance**, John Wiley & Sons, Inc., New York, NY, USA, v. 7, n. 1, p. 49–62, jan. 1995.

WONG, W. E.; GOKHALE, S. S.; HORGAN, J. R. Quantifying the closeness between program components and features. **Journal of Systems and Software**, Elsevier Science Inc., New York, NY, USA, v. 54, n. 2, p. 87–98, out. 2000.

YE, Y.; FISCHER, G.; REEVES, B. Integrating active information delivery and reuse repository systems. **SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT'2000)**, ACM, New York, NY, USA, v. 25, n. 6, p. 60–68, nov. 2000. ISSN 0163-5948.

Anexos

Anexo 1

Trechos do Livro de Receitas do Framework JHotDraw.

Cookbook

Framework JHotDraw

Sobre o Cookbook:

Um framework orientado a objetos utiliza técnicas típicas de orientação à objetos para incorporar características flexíveis, os ***hot-spots são classes e métodos do framework que****devem ser usados de forma adequada para integrar as necessidades de aplicações específicas ao framework. Este cookbook apresenta um conjunto de Receitas para instanciar funcionalidades usando o framework JHotDraw, versão 5.3. Esta é uma documentação criada com base em exemplos, logo as sugestões de atividades e exemplos são retirados de uma ou mais aplicações exemplo e suas* ***classes específicas****, criadas para esta aplicação e que usam algum hot-spot.*

As Receitas apresentam informações sobre os hot-spots e exemplos de utilização destes na instanciação desta funcionalidade para outra aplicação. Segue o tipo de informação que será apresentada.

*1.****Uma lista de Atividades*** *como Extender, Implementar e Usar um hot-spot do Framework para a execução da Receita.*

i. Comentários de Código: *Comentário de código sobre a Classe hot-spot que pode ser usada.*

ii. Assinatura da Classe hot spot.

iii. Padrão de Projeto: *Padrão de Projeto que deve ser seguido para usar a classe hot-spot. Pode não seguir padrão de projeto, então este campo será vazio.*

iv. Metodos da classe hot-spot a serem possivelmente redefinidos (overriden) para a receita selecionada.

a. Classe específica da aplicação exemplo que "extends" a classe hot-spot.

b. Nome do Método hotspot.

c. Parâmetros deste método hot-spot.

d. Código do Método hot-spot na Classe hot-spot.

e. Comentário de Código do Método hot-spot na Classe hot-spot

f. Exemplos de redefinição do método: *Apresenta um exemplo da redefinição do método na classe específica da aplicação exemplo, que usa o framework.*

g. Comentários de Código do exemplo: *Comentário de código da redefinição do método na aplicação exemplo.*

v. Instanciação de objetos das classes hot-spot

a. Classe específica da aplicação exemplo em que o objeto da classe hot-spot é instanciado.

b. Comentário de código da Classe específica.

c. Exemplos de instanciação do objeto : *Apresenta um exemplo de instanciação do objeto da classe hotspot na aplicação exemplo.*

d. Comentários de Código do exemplo: *Comentário de código da instanciação do objeto da classe hotspot na aplicação exemplo.*

vi. Métricas do framework: *A seguir são apresentadas as métricas de código do framework JHotDraw.*

Lista de Receitas:

A seguir são apresentadas as Receitas deste Cookbook. As informações de uma ou mais receitas PODEM ser uteis para realizar a atividade alvo.

Nome da Receita	Descrição da Receita
Inicializar o Sistema - JHotDraw App	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para conhecer como os menus e funcionalidades da aplicação são carregadas e apresentadas na aplicação JHotDraw. Esta receita foi obitida por meio da inicialização da aplicação JHotDraw, que usa o framework de

	mesmo nome.
Inicializar Sistema - JModeller	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para conhecer como os menus e funcionalidades da aplicação são carregadas e apresentadas na aplicação JModeller. Esta receita foi obitida por meio da inicialização da aplicação JModeller, que usa o framework JHotDraw.
New - Nova Imagem	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a abertura/criação de uma nova janela para criar um novo desenho.
Selecionar (clicar) Botão para Desenhar Figura Elipse	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para seleção de uma das figuras presente no Menu principal da aplicação exemplo. Apenas a seleção, clicar no botão para selecionar uma figura, que neste exemplo é a figura Elipse. Ao selecionar uma figura, a ferramenta irá possibilitar desenhar tal figura.
Desenhar Figura Elipse	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar a figura Elipse e outras figuras. A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar esta figura.
Desenhar Figura Losango	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar a figura Losango e outras figuras. A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar esta figura.
Colorir Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para colorir uma figura, mudar a cor da mesma. Neste caso coletamos informações sobre como colorir a figura quadrado
Colorir Borda da Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis colorir a borda (linha que delimita) uma figura, mudar a cor da borda. Neste caso coletamos informações sobre como colorir a borda de uma figura quadrada.
Copiar (Copy) Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por criar uma copia da figura selecionada.
Colar (Paste) Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por “colar”, adicionar na imagem uma copia da figura copiada anteriormente. A receita anterior trata da funcionalidade de realizar a cópia da figura.

Cortar (Cut)Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por “cortar”, remover uma figura da imagem e copiar a mesma para ser futuramente adicionada (“colada”) à imagem novamente. A receita anterior trata da funcionalidade de realizar a adição da figura.
Duplicar Figura (Duplicate Figure)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável duplicar figura selecionada, ou seja fazer uma cópia e colar na imagem.
Mover (Move) Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por “Mover” uma figura, selecionar (clicar) e “arrastar” a figura para alterar a posição da figura na imagem. Neste caso coletamos informações sobre uma figura quadrada.
Desfazer (Undo) Mover Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por desfazer (“undo”) uma ação. Neste caso coletamos informações sobre o desfazer a ação de mover uma figura quadrada.
Refazer (Redo) Mover Figura	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por refazer (“redo”) uma ação que foi anteriormente desfeita. A receita anterior trata da funcionalidade de desfazer uma ação. Neste caso coletamos informações sobre refazer a ação ligada a movimentação de uma figura quadrada.
Alinhar Figuras (Align Command)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por alinhar (“Align”) duas ou mais figuras selecionadas na imagem.
Alinhar Figuras ao Centro (Align Command Center)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por alinhar ao centro da imagem (“Align Command Center”) duas ou mais figuras selecionadas.
Trazer para frente (Bring to Front)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por trazer figura selecionada uma camada para frente (“bring to front") na imagem.
Enviar para Trás (Send to Back)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por trazer figura selecionada uma camada para atrás (“bring to back”) na imagem.
Agrupar Figuras (Group)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por agrupar (“group) uma ou mais figuras selecionadas.

Desagrupar Figuras (UnGroup)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade responsável por separar uma ou mais figuras agrupadas (“Ungroup) selecionadas.
Selecionar Elbow Connection Tool	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para seleção do conector "cotovelo" (Elbow Connection) no Menu principal da aplicação exemplo (apenas a seleção, clicar no botão para seleciona-lo). Ao selecionar, a ferramenta irá possibilitar desenhar tal conector.
Desenhar Elbow Connection Tool	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar uma conector "cotovelo" (Elbow Connection). A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar este conector. Esta receita foi obitida com por meio da conecção entre duas figuras retangulares.
Alterar a forma da Elbow Connection	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para alterar a forma do conector "cotovelo" . Esta é um conector com o formato de "cotovelo" que pode ter suas arestas alteradas.
Selecionar Line Connection Tool	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para seleção da linha conectora no Menu principal da aplicação exemplo (apenas a seleção, clicar no botão para seleciona-la). Ao selecionar, a ferramenta irá possibilitar desenhar tal conector.
Desenhar Line Connection Tool	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar uma linha que conecta duas figuras (line tool). A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar esta linha conectora. Esta receita foi obitida com por meio da conecção entre duas figuras retangulares.
Selecionar Seta de Heranca - JModeller	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para seleção da Seta de "Herança" conectora (seta conectora com um triangulo na ponta) no Menu principal da aplicação exemplo (apenas a seleção, clicar no botão para seleciona-la). Ao selecionar, a ferramenta irá possibilitar desenhar tal conector.
Desenhar Seta de Heranca (seta com triangulo na ponta) - JModeller	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar uma Seta de "Herança" conectora (seta conectora com um triangulo na ponta). A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar esta seta conectora. Esta receita foi obitida com por meio da conecção entre duas figuras retangulares.
Seta em ambos os Lados do	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade que acresccenta setas em ambas as extermidades de conector já inserida ne imagem conectando duas figuras. Esta receita foi obtida utilizando o exemplo da Linha Conectora entre dois

Conector (Arrow Both)	retangulos.
Seta no final do Conector (Arrow at End)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade que acresccenta uma seta no final de um conector (apontando para a figura que recebeu o conector) já inserida ne imagem conectando duas figuras. Esta receita foi obtida utilizando o exemplo da Linha Conectora entre dois retangulos.
Seta no início do Conector (Arrow at Start)	Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade que acresccenta uma seta no início de um conector (apontando para a figura de onde partiu o conector) já inserida ne imagem conectando duas figuras. Esta receita foi obtida utilizando o exemplo da Linha Conectora entre dois retangulos.

voltar

Receita: Desenhar Figura Losango

Descrição:

Esta receita apresenta informações sobre os hot spots do framework que podem ser úteis para a funcionalidade desenhar a figura Losango e outras figuras. A seleção da figura no menu foi contemplada na receita anterior, esta receita trata apenas do ato de desenhar esta figura.

Lista de Classes Hot-spot:

A seguir são apresentadas as Classes hot-spots do framework que podem ser usadas na aplicação exemplo para as funcionalidades tratadas por esta receita. Estas PODEM ser uteis para a sua atividade.

Estender Classe Hot-spot (extends Classe Hot-spot)

A seguir são apresentadas as Classes hot-spots do framework que PODEM ser estendidas ("extends") na aplicação exemplo para as funcionalidades tratadas por esta receita.

Hotspot Class (on trace)	Comentário de Código da Classe Hotspot
CH.ifa.draw.standard.DecoratorFigure	/** * DecoratorFigure can be used to decorate other figures with * decorations like borders. Decorator forwards all the * methods to their contained figure. Subclasses can selectively * override these methods to extend and filter their behavior. * <hr> * Design Patterns<P> * <img src="images/red-ball-small.gif" width=6 height=6 alt=" o "> * Decorator * DecoratorFigure is a decorator. * * @see Figure * * @version <$CURRENT_VERSION$> */
CH.ifa.draw.standard.StandardDrawing	/** * The standard implementation of the Drawing interface. * * @see Drawing * * @version <$CURRENT_VERSION$> */

Colaborar com Classe Hotspot

A seguir são apresentadas as Classes hot-spots do framework que PODEM ser usadas (ex: instanciação de objeto) na aplicação exemplo para as funcionalidades tratadas por esta receita.

Hotspot Class (on trace)	Comentário de Código da Classe Hotspot
CH.ifa.draw.contrib.DiamondFigure	/** * A diamond with vertices at the midpoints of its enclosing rectangle * * @author Doug Lea (dl at gee, Tue Feb 25 17:39:44 1997) * @version <$CURRENT_VERSION$> */
CH.ifa.draw.contrib.WindowMenu	/** * Menu component that handles the functionality expected of a standard * "Windows" menu for MDI applications. * * @author Wolfram Kaiser (adapted from an article in JavaWorld) * @version <$CURRENT_VERSION$> */
CH.ifa.draw.standard.CreationTool	/** * A tool to create new figures. The figure to be * created is specified by a prototype. * * <hr> * Design Patterns<P> * <img src="images/red-ball-small.gif" width=6 height=6 alt=" o "> * Prototype * CreationTool creates new figures by cloning a prototype. * <hr> * * @see Figure * @see Object#clone * * @version <$CURRENT_VERSION$> */
CH.ifa.draw.standard.ToolButton	/** * A PaletteButton that is associated with a tool. * * @see Tool * * @version <$CURRENT_VERSION$> */
CH.ifa.draw.util.CommandMenu	/** * A Command enabled menu. Selecting a menu item * executes the corresponding command. * * @see Command * * @version <$CURRENT_VERSION$> */
CH.ifa.draw.util.UndoableCommand	/** * @author Wolfram Kaiser * @version <$CURRENT_VERSION$> */

CH.ifa.draw.util.UndoableTool	/** * @author Wolfram Kaiser * @version <$CURRENT_VERSION$> */
CH.ifa.draw.figures.RectangleFigure	/** * A rectangle figure. * * @version <$CURRENT_VERSION$> */

Reuso na Hierarquia de Super Classe

A seguir são apresentadas as Classes hot-spots do framework que PODEM ser usadas por herança (ex: invocação de algum método de super classe hot-spot) na aplicação exemplo para as funcionalidades tratadas por esta receita. Esta lista contempla casos como: A extends B e B extends C, logo A pode usar métodos de C.

Hotspot Class (on trace)	Comentário de Código da Classe Hotspot

CH.ifa.draw.application.DrawApplication	/**
	* DrawApplication defines a standard presentation for
	* standalone drawing editors. The presentation is
	* customized in subclasses.
	* The application is started as follows:
	* <pre>
	* public static void main(String[] args) {
	* MayDrawApp window = new MyDrawApp();
	* window.open();
	* }
	* </pre>
	*
	* @version <$CURRENT_VERSION$>
	*/

Exemplo de Uso dos Hot-spots:

A seguir são apresentados exemplos de Classes Específicas de aplicações que usam os Hotspots (do framework) apresentados para esta receita. Estas PODEM servir de EXEMPLO para a sua atividade.

Classe Específica - Exemplo de uso	Classe Hotspot	Relacionamento	Código da Classe Específica
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.contrib.DiamondFigure	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.application.DrawApplication	Reuso na Hierarquia de Super Classe	

CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.contrib.WindowMenu	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.standard.CreationTool	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.standard.ToolButton	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.util.CommandMenu	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.util.UndoableCommand	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.util.UndoableTool	Colaboração	
CH.ifa.draw.samples.javadraw.JavaDrawApp	CH.ifa.draw.figures.RectangleFigure	Colaboração	
CH.ifa.draw.samples.javadraw.AnimationDecorator	CH.ifa.draw.standard.DecoratorFigure	Estender Hotspot	
CH.ifa.draw.samples.javadraw.BouncingDrawing	CH.ifa.draw.standard.StandardDrawing	Estender Hotspot	

Classe Hot-Spot

CH.ifa.draw.standard.DecoratorFigure

A seguir serão apresentadas informações e exemplos de uso deste hot-spot para a(s) funcionalidade(s) tratada(s) pela Receita selecionada.

Comentario de Código:

```
/**
* DecoratorFigure can be used to decorate other figures with
* decorations like borders. Decorator forwards all the
* methods to their contained figure. Subclasses can selectively
* override these methods to extend and filter their behavior.
* <hr>
* <b>Design Patterns</b><P>
* <img src="images/red-ball-small.gif" width=6 height=6 alt=" o ">
* <b><a href=../pattlets/sld014.htm>Decorator</a></b><br>
* DecoratorFigure is a decorator.
*
* @see Figure
*
* @version <$CURRENT_VERSION$>
*/
```

Assinatura da Classe:

```
public abstract class DecoratorFigure
extends AbstractFigure
implements FigureChangeListener {
```

Padrão de Projeto

DESIGN PATTERN		
Decorator	Component: CH.ifa.draw.framework.Figure	Decorator: CH.ifa.draw.standard.DecoratorFigure component: Figure fComponent Operation(): displayBox, basicDisplayBox, draw, handles

Extends Classe Hot-spot

A seguir serão apresentadas classes específicas da aplicação exemplo que "extends" essa classe Hot-spot e podem ser uteis para a receita.

Você PODE ter que criar ou alterar uma classe da aplicação que extends CH.ifa.draw.standard.DecoratorFigurecom no exemplo:

CH.ifa.draw.samples.javadraw.AnimationDecorator extends CH.ifa.draw.standard.DecoratorFigure 

Redefinições e Invocações de Métodos do Hotspot

A seguir serão apresentadas as redefinições e invocações de métodos hot-spot desta classe hot-spot que podem ser uteis para esta Receita.

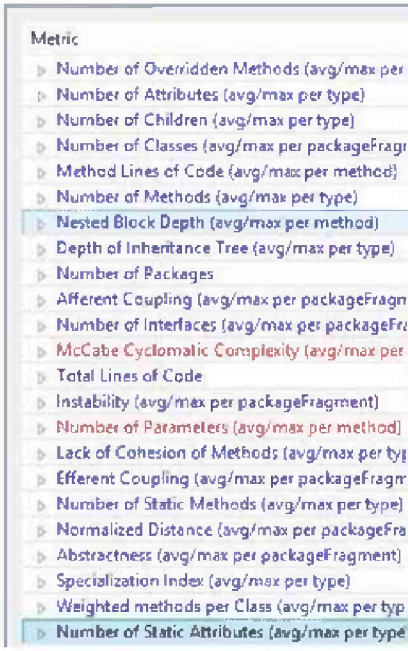
Você pode ter que redefinir métodos na classe da aplicação que "extends" CH.ifa.draw.standard.DecoratorFigure.
Exemplo: CH.ifa.draw.samples.javadraw.AnimationDecorator extends CH.ifa.draw.standard.DecoratorFigure

Você pode ter que redigir os seguintes métodos para esta receita:

Metodo	Parâmentro	Comentário de Código no Hotspot	Exemplo de Redefinição do Método	Exemplo de Invocação do Método
basicDisplayBox	(Point origin, Point corner)	/** * Forwards basicDisplayBox to its contained figure. */		

Métricas do Framework JFace e SWT:

A seguir são apresentadas as métricas de código do framework JFace e SWT.


Metric
Number of Overridden Methods (avg/max per
Number of Attributes (avg/max per type)
Number of Children (avg/max per type)
Number of Classes (avg/max per packageFragr
Method Lines of Code (avg/max per method)
Number of Methods (avg/max per type)
Nested Block Depth (avg/max per method)
Depth of Inheritance Tree (avg/max per type)
Number of Packages
Afferent Coupling (avg/max per packageFragn
Number of Interfaces (avg/max per packageFr
McCabe Cyclomatic Complexity (avg/max per
Total Lines of Code
Instability (avg/max per packageFragment)
Number of Parameters (avg/max per method)
Lack of Cohesion of Methods (avg/max per typ
Efferent Coupling (avg/max per packageFragm
Number of Static Methods (avg/max per type)
Normalized Distance (avg/max per packageFra
Abstractness (avg/max per packageFragment)
Specialization Index (avg/max per type)
Weighted methods per Class (avg/max per typ
Number of Static Attributes (avg/max per type

Total	Mean	Std. Dev.	Maxim...
260	1.232	1.569	12
383	1.815	2.363	17
160	0.758	2.778	26
211	19.182	24.233	82
8494	4.243	6.833	101
1924	9.118	10.772	72
	1.366	0.699	5
	2.654	1.649	8
11			
	33.636	50.14	149
33	3	5.427	18
	1.637	1.679	38
14864			
	0.583	0.373	1
	0.958	1.088	8
	0.266	0.353	1.5
	14.636	16.244	55
78	0.37	1.642	16
	0.285	0.311	0.857
	0.132	0.232	0.818
	0.513	0.656	3
3277	15.531	20.387	146
108	0.512	1.09	8

Classe Hot-Spot

CH.ifa.draw.application.DrawApplication

A seguir serão apresentadas informações e exemplos de uso deste hot-spot para a(s) funcionalidade(s) tratada(s) pela Receita selecionada.

Comentario de Código:

```
/**
* DrawApplication defines a standard presentation for
* standalone drawing editors. The presentation is
* customized in subclasses.
* The application is started as follows:
* <pre>
* public static void main(String[] args) {
* MayDrawApp window = new MyDrawApp();
* window.open();
* }
* </pre>
*
* @version <$CURRENT_VERSION$>
*/
```

Assinatura da Classe:

```
public class DrawApplication
extends JFrame
implements DrawingEditor, PaletteListener, VersionRequester {
```

Padrão de Projeto

DESIGN PATTERN		
Object Adapter- Command	Adapter/ConcreteCommand: CH.ifa.draw.application.DrawApplication adaptee: ToolButton fDefaultToolButton Request(): toolDone Target role is played by class CH.ifa.draw.framework.DrawingEditor adaptee: ToolButton fSelectedToolButton Request(): paletteUserOver Target role is played by class CH.ifa.draw.util.PaletteListener	Adaptee/Receiver: CH.ifa.draw.standard.ToolButton SpecificRequest(): name, tool
State-Strategy	Context: CH.ifa.draw.applet.DrawApplet CH.ifa.draw.application.DrawApplication	State/Strategy: CH.ifa.draw.framework.Tool

State-Strategy	Context: CH.ifa.draw.applet.DrawApplet CH.ifa.draw.application.DrawApplication	State/Strategy: CH.ifa.draw.framework.Drawing

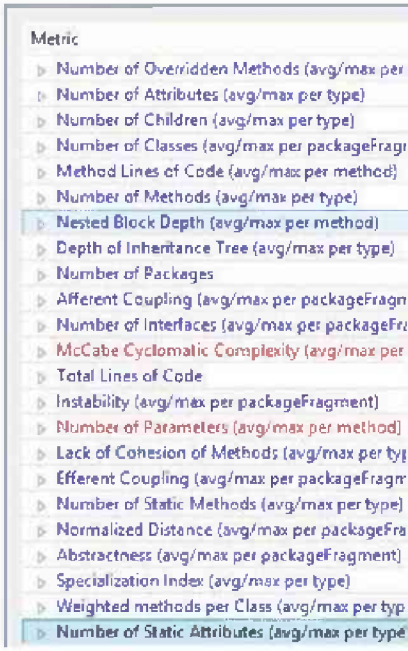
Instanciações do Hotspot

A seguir serão apresentadas as instanciações de objetos desta classe hot-spot.

Classe Especifica da Aplicação Exemplo	Comentário de Código da Classe Especifica	Exemplo de Instanciação do Objeto na Classe Especifica	Comentario de Código do Exemplo
CH.ifa.draw.samples.javadraw.JavaDrawApp	/** * @version <$CURRENT_VERSION$> */		/** * Factory method which create a new instance of this * application. * * @return newly created application */

Métricas do Framework JFace e SWT:

A seguir são apresentadas as métricas de código do framework JFace e SWT.


Metric
Number of Overridden Methods (avg/max per
Number of Attributes (avg/max per type)
Number of Children (avg/max per type)
Number of Classes (avg/max per packageFragr
Method Lines of Code (avg/max per method)
Number of Methods (avg/max per type)
Nested Block Depth (avg/max per method)
Depth of Inheritance Tree (avg/max per type)
Number of Packages
Afferent Coupling (avg/max per packageFragn
Number of Interfaces (avg/max per packageFr
McCabe Cyclomatic Complexity (avg/max per
Total Lines of Code
Instability (avg/max per packageFragment)
Number of Parameters (avg/max per method)
Lack of Cohesion of Methods (avg/max per typ
Efferent Coupling (avg/max per packageFragm
Number of Static Methods (avg/max per type)
Normalized Distance (avg/max per packageFra
Abstractness (avg/max per packageFragment)
Specialization Index (avg/max per type)
Weighted methods per Class (avg/max per typ
Number of Static Attributes (avg/max per type

Total	Mean	Std. Dev.	Maxim...
260	1.232	1.569	12
383	1.815	2.363	17
160	0.758	2.778	26
211	19.182	24.233	82
8494	4.243	6.833	101
1924	9.118	10.772	72
	1.366	0.699	5
	2.654	1.649	8
11			
	33.636	50.14	149
33	3	5.427	18
	1.637	1.679	38
14864			
	0.583	0.373	1
	0.958	1.088	8
	0.266	0.353	1.5
	14.636	16.244	55
78	0.37	1.642	16
	0.285	0.311	0.857
	0.132	0.232	0.818
	0.513	0.656	3
3277	15.531	20.387	146
108	0.512	1.09	8

Anexo 2

Questionário de Seleção dos Profissionais.

Questionário de Seleção dos Alunos.

Questionário 1 - Seleção Participantes

Sobre a Pesquisa:

Você está sendo convidado (a) para participar da pesquisa intitulada **INSTANCIAÇÃO DE FRAMEWORKS USANDO COOKBOOKS CONSTRUÍDOS A PARTIR ANÁLISE ESTÁTICA E ANÁLISE DINÂMICA DE CÓDIGO-FONTE.** Sob a responsabilidade dos pesquisadores **Marcelo de Almeida Maia e Raquel Fialho de Queiroz Lafetá.**

Nesta pesquisa buscamos verificar os benefícios de um nova forma de documentação de frameworks, baseada no código fonte. Pretendemos verificar se o documento propostos são uma alternativa viável de documentação, isto é, que seja tão bom ou melhor que documentação gerada manualmente, em termos de taxa de acerto e tempo de execução da atividade. Para este fim, será necessário comparar o desempenho de desenvolvedores (participantes da pesquisa) utilizando as diferentes documentações.

Se aceitar participar, você irá realizar atividades reais de instanciação de frameworks que utilizam a linguagem JAVA, onde será necessária a compreensão da instanciação e modificações no código. Cada participante irá executar no mínimo duas atividades de instanciação. Todas as dúvidas referentes ao projeto de pesquisa serão esclarecidas. **Os voluntários estarão livres para retirar o seu consentimento de participação na pesquisa sem nenhum prejuízo ou coação.**

A equipe responsável pelo estudo se compromete a **manter em sigilo absoluto a identidade dos sujeitos participantes incluídos no estudo.** Para mais informações, não se esqueça de ler e assinar o TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO que será enviado por e-mail.

Caso você aceite participar, seguem algumas perguntas que devem ser respondidas para a seleção dos participantes. Você não irá gastas mais do **10 minutos** respondendo à este questionário.

Obrigada pela colaboração!

Raquel Fialho

Email: raquel.rafialho@gmail.com

Skype: rafialho

1 Dados do Participante:

Nome

Empresa que trabalha

Instituição onde estuda ou estudou

Idade

Cidade/Município

Estado

Formação Acadêmica

Endereço de email

2 Maior grau de escolaridade

- ◯ Graduação
- ◯ Especialização
- ◯ Mestrado
- ◯ Doutorado
- ◯ Pós-doutorado

Em qual instituição?

3 Anos de experiência profissional (contar com estágio)

- ◯ meses
- ◯ 1 - 3 anos
- ◯ 4 - 6 anos
- ◯ 7 - 9 anos
- ◯ 10 - 13 anos
- ◯ 13 - 15 anos
- ◯ 16 - 17 anos
- ◯ 18 - 20 anos
- ◯ 21 - 29 anos
- ◯ Mais de 30 anos

4 Qual é a sua área de atuação em TI (pode marcar mais de uma opção)

- [ ] Arquitetura de software
- [ ] Banco de Dados
- [ ] Business Intelligence
- [ ] Desenvolvedor Java
- [ ] Desenvolvedor Dot.Net
- [ ] Desenvolvedor PL-Sql
- [ ] Desenvolvedor - Outros
- [ ] Engenharia Reversa
- [ ] Engenharia de Software
- [ ] Gestor de Projetos
- [ ] Governança
- [ ] Infraestrutura e/ou Redes
- [ ] Qualidade de Software
- [ ] Modelagem de Software
- [ ] Segurança da Informação
- [ ] Sistemas Operacionais
- [ ] Teste de Software
- [ ] Tecnologia voltada aos negócios

Outro (especifique)

5 Sobre orientação à objetos (OO) e linguagem Java. Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ Outro. Use este campo caso nunca tenha trabalhado com Java.

6 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre o uso do IDE Eclipse:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ Outro (especifique). Selecione esta opção, caso nunca tenha usado o IDE.

7 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre como usar frameworks:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

Quantos (número) frameworks você já usou, aproximadamente?

8 Quais dos Frameworks abaixo você já usou? Marque os frameworks usados.

- ☐ Android framework
- ☐ Eclipse Plugins Framework
- ☐ Java Media Framework
- ☐ JHotDraw
- ☐ Swing
- ☐ Spring
- ☐ Jface
- ☐ OUTROS, liste os frameworks mais usados por você:

9 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre Padrões de Projeto:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

Quais Padrões de Projeto você já usou?

Obrigada pela sua participação!

Questionário 1 - Seleção Participantes - Alunos FeMASS

Sobre a Pesquisa:

Você está sendo convidado (a) para participar da pesquisa intitulada **INSTANCIAÇÃO DE FRAMEWORKS USANDO COOKBOOKS CONSTRUÍDOS A PARTIR ANÁLISE ESTÁTICA E ANÁLISE DINÂMICA DE CÓDIGO-FONTE.** Sob a responsabilidade dos pesquisadores **Marcelo de Almeida Maia e Raquel Fialho de Queiroz Lafetá.**

Nesta pesquisa buscamos verificar os benefícios de um nova forma de documentação de frameworks, baseada no código fonte. Pretendemos verificar se o documento propostos são uma alternativa viável de documentação, isto é, que seja tão bom ou melhor que documentação gerada manualmente, em termos de taxa de acerto e tempo de execução da atividade. Para este fim, será necessário comparar o desempenho de desenvolvedores (participantes da pesquisa) utilizando as diferentes documentações.

Se aceitar participar, você irá realizar atividades reais de instanciação de frameworks que utilizam a linguagem JAVA, onde será necessária a compreensão da instanciação e modificações no código. Cada participante irá executar no mínimo duas atividades de instanciação. Todas as dúvidas referentes ao projeto de pesquisa serão esclarecidas. **Os voluntários estarão livres para retirar o seu consentimento de participação na pesquisa sem nenhum prejuízo ou coação.**

A equipe responsável pelo estudo se compromete a **manter em sigilo absoluto a identidade dos sujeitos participantes incluídos no estudo.** Para mais informações, não se esqueça de ler e assinar o TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO que será enviado por e-mail.

Caso você aceite participar, seguem algumas perguntas que devem ser respondidas para a seleção dos participantes. Você não irá gastas mais do **5 minutos** respondendo à este questionário.

Obrigada pela colaboração!

Raquel Fialho

Email: raquel.rafialho@gmail.com

Skype: rafialho

1 Dados do Participante:

Nome	
Período (faculdade)	
Instituição onde estuda ou estudou	
Idade	
Cidade/Município	
Estado	
Endereço de email	

2 Você trabalha ou já trabalhou com desenvolvimento de Software?

- ◯ NÃO
- ◯ SIM (Por quanto tempo?)

3 Quantas aplicações em Java você já desenvolveu (podem ser trabalhos da faculdade)?

4 Quantas aplicações (em qualquer linguagem) você já desenvolveu (podem ser trabalhos da faculdade)?

5 Você cursou ou está cursando a disciplina sobre programação OO / Java?

- ◯ SIM
- ◯ NÃO

6 Você cursou ou cursa disciplina sobre frameworks?

- ◯ SIM
- ◯ NÃO

7 Você já utilizou o IDE Eclipse anteriormente?

- ◯ SIM
- ◯ NÃO

8 Você já desenvolveu algum plugin do Eclipse?

- ◯ NÃO
- ◯ SIM (Liste quantos)

9 Sobre orientação à objetos (OO) e linguagem Java. Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ Outro. Use este campo caso nunca tenha trabalhado com Java.

10 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre o uso do IDE Eclipse:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5
- ◯ Outro (especifique). Selecione esta opção, caso nunca tenha usado o IDE.

11 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre como usar frameworks:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

Quantos (número) frameworks você já usou, aproximadamente?

12 Quais dos Frameworks abaixo você já usou? Marque os frameworks usados.

- [ ] Android framework
- [ ] Eclipse Plugins Framework
- [ ] JHotDraw
- [ ] Swing
- [ ] Spring
- [ ] Jface
- [ ] SWT - Standard Widget Toolkit
- [ ] Nenhum dos anteriores
- [ ] OUTROS, liste os frameworks mais usados por você:

13 Numa escala de 1 à 5, sendo 5 a maior nota, como você considera o seu conhecimentos sobre Padrões de Projeto:

- ◯ 1
- ◯ 2
- ◯ 3
- ◯ 4
- ◯ 5

Quais Padrões de Projeto você já usou?

Obrigada pela sua participação!

Anexo 3

Questionário de Coleta dos Dados - Grupo Controle.

Questionário - Coleta de Dados - ATIVIDADE 2

Instruções para responder o questionário

PRIMEIRAMENTE, OBRIGADA!

LEIA O E_MAIL COM INSTRUÇÕES SOBRE A ATIVIDADE ANTES DE RESPONDER ESTE QUESTIONÁRIO.

Este questionário deve ser respondido depois que a atividade solicitada seja realizada. Por favor, sejam sinceros em suas respostas. Este apresenta 15 perguntas simples para avaliar a documentação utilizada durante a instanciação e sua percepção sobre a execução da atividade. Responda todas as perguntas.

Você vai precisar saber o que são classes e métodos hot-spot:
Um framework orientado a objetos utiliza técnicas típicas de orientação à objetos para incorporar características flexíveis, os hot-spots são classes e métodos do framework que devem ser usados de forma adequada para integrar as necessidades de aplicações específicas ao framework.

LEMBRETE: Envie o código do sistema com as alterações solicitadas, compactado, para raquel.rafialho@gmail.com.

Este questionário que deve ser respondido depois que a atividade solicitada seja realizada. Por favor, sejam sinceros em suas respostas. Este apresenta 27 perguntas simples para avaliar a documentação utilizada durante a instanciação e sua percepção sobre a execução da atividade. Responda todas as perguntas.

Você vai precisar saber o que são classes e métodos hot-spot:
Um framework orientado a objetos utiliza técnicas típicas de orientação à objetos para incorporar características flexíveis, os hot-spots são classes e métodos do framework que devem ser usados de forma adequada para integrar as necessidades de aplicações específicas ao framework.

LEMBRETE: Envie o código do sistema com as alterações solicitadas, compactado, para raquel.rafialho@gmail.com.

Por favor, leia o Termo de Consentimento abaixo. Este termo esclarece sobre a sua participação. Se aceito, você estará confirmando a sua participação na pesquisa.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Você está sendo convidado (a) para participar da pesquisa intitulada INSTANCIAÇÃO DE FRAMEWORKS USANDO COOKBOOKS CONSTRUÍDOS A PARTIR ANÁLISE ESTÁTICA E ANÁLISE DINÂMICA DE CÓDIGO-FONTE. sob a responsabilidade dos pesquisadores Marcelo de Almeida Maia e Raquel Fialho de Queiroz Lafetá. Nesta pesquisa buscamos verificar os benefícios de um nova forma de documentação de frameworks, baseada no código fonte.

Na sua participação você irá realizar atividades reais de instanciação de frameworks que utilizam a linguagem JAVA, onde será necessária a compreensão da instanciação e modificações no código. Cada participante irá executar no mínimo duas atividades de instanciação. Estas atividades e as instruções de execução serão passadas pela pesquisadora. O tempo de execução e código gerado será coletado e avaliado. Ao final da execução da atividade, você irá responder a um questionário sobre a atividade e o documento utilizado.

Os riscos destes experimentos consistem em: você poderá se sentir cansado com a atividade proposta, considerado semelhante ao risco de ficar desanimado com atividades que são habituais (do dia a dia) aos analistas. Poderá se sentir frustrado se não conseguir responder algum item do questionário. Também, existe o risco de o participante ser identificado. Porém, a pesquisa foi estruturada para que em nenhum momento você seja identificado. Os resultados da pesquisa serão publicados e ainda assim a sua identidade será preservada, visando não causar desconforto com relação aos resultados e respostas expostas. Apenas os pesquisadores envolvidos no estudo poderão avaliar as informações. A aplicação dos questionários propostos será conduzida de forma a não induzir, coibir, constranger ou reprimir qualquer comportamento ou informação relatada. Você não terá nenhum gasto e ganho financeiro por participar na pesquisa.

Os benefícios serão, ao final deste estudo, as informações geradas poderão trazer benefícios à você e outros analistas. Por meio das informações coletadas e analisadas serão gerados resultados compilados que podem auxiliar futuramente nas documentações de frameworks que são amplamente utilizados pelos analistas de TI em geral, atingindo diretamente ou indiretamente os sujeitos participantes.

Todas as dúvidas referentes ao projeto de pesquisa serão esclarecidas. Os voluntários estarão livres para retirar o seu consentimento de participação no projeto sem nenhum prejuízo ou coação. Uma via original deste Termo de Consentimento Livre e Esclarecido ficará com você. Qualquer dúvida a respeito da pesquisa, você poderá entrar em contato com a pesquisadora Raquel Fialho pelo contato: raquel.rafialho@gmail.com, skype: rafialho ou entrar em contato com Comitê de Ética na Pesquisa com Seres-Humanos – Universidade Federal de Uberlândia, fone: 34-32394131.

1. Você aceita participar do projeto citado acima, voluntariamente, após ter sido devidamente esclarecido?

- ◯ SIM
- ◯ NÃO

2. Seu Nome:

* 3. Marque abaixo o horário de Início e Fim de execução da Atividade:

Início da Atividade (hh:mm:ss):

Fim da Atividade (hh:mm:ss):

4. Se necessário realizar uma pausa, marque abaixo o horário de Início e Fim da(s) pausa(s).

Início da Pausa 1 (hh:mm:ss):

Fim da Pausa 1 (hh:mm:ss):

Início da Pausa 2 (hh:mm:ss):

Fim da Pausa 2 (hh:mm:ss):

5. Você concluiu a atividade com sucesso?

- ◯ SIM
- ◯ NÃO
- ◯ Parcialmente (Justificar):

* 6. O mecanismo de navegação pelo documento foi útil para a atividade? Marque o nível de utilidade conforme:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

7. Sobre as informações apresentadas na Documentação, qual foi a utilidade destas para a execução da Atividade? Marque o nível de utilidade conforme:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justifique esta resposta:

8. Você usou alguma Classe Hot-spot do framework ("extends", "implements" ou instanciação da classe), qual(is) e como?

- ◯ Não
- ◯ SIM, liste a(s) classe(s):

9. Faltou alguma Classe Hot-spot (do framework) importante para atividade na Documentação enviada?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais classes faltaram e por que (para que) as usou:

10. Na documentação enviada, faltou alguma informação importante sobre alguma Classe Hot-spot do framework para execução da atividade ?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais informações faltaram e por que (para que) as usou:

11. Você redefiniu algum Método Hot-spot do framework?

- ◯ NÃO
- ◯ SIM, liste o(s) método(s).

12. Você invocou algum Método Hot-spot do framework?

- ◯ Não
- ◯ Sim (liste os métodos invocados)

13. Na documentação enviada, faltou algum Método Hot-spot do framework importante para atividade?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais métodos faltaram e por que (para que) os usou:

14. Na documentação enviada, faltou alguma informação importante sobre algum Método Hot-spot para execução da atividade?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais informações faltaram e por que (para que) as usou:

15. As informações sobre MÉTRICAS do framework JHotDraw 5.3 foram úteis para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

16. Esta atividade foi COMPLEXA de realizar?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justifique (OPCIONAL):

17. Você ficou SATISFEITO com o uso do Documento para realizar esta atividade?

Satisfação: Sensação agradável que sentimos quando as coisas correm à nossa vontade ou se cumprem a nosso contento.

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa:

18. Foi FÁCIL LOCALIZAR as informações necessárias para instanciação no Documento?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

19. As informações dispostas no documento estavam CLARAS, ou seja, facilmente compreensíveis e bem apresentadas?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

Anexo 4

Questionário de Coleta dos Dados - Grupo Receitas.

Questionário - Coleta de Dados Cookbook - Atividade 1

Instruções para responder o questionário

Este questionário que deve ser respondido depois que a atividade solicitada seja realizada. Por favor, sejam sinceros em suas respostas. Este apresenta 26 perguntas simples para avaliar a documentação utilizada durante a instanciação e sua percepção sobre a execução da atividade. Responda todas as perguntas.

Você vai precisar saber o que são classes e métodos hot-spot:
Um framework orientado a objetos utiliza técnicas típicas de orientação à objetos para incorporar características flexíveis, os hot-spots são classes e métodos do framework que devem ser usados de forma adequada para integrar as necessidades de aplicações específicas ao framework.

LEMBRETE: Envie o código do sistema com as alterações solicitadas, compactado, para raquel.rafialho@gmail.com.

1. Seu Nome:

* 2. Digite abaixo o horário de Início e Fim de execução da Atividade:

Início da Atividade (hh:mm:ss):

Fim da Atividade (hh:mm:ss):

3. Se necessário realizar uma pausa, marque abaixo o horário de Início e Fim da(s) pausa(s).

Início da Pausa 1 (hh:mm:ss):

Fim da Pausa 1 (hh:mm:ss):

Início da Pausa 2 (hh:mm:ss):

Fim da Pausa 2 (hh:mm:ss):

4. Você concluiu a atividade com sucesso?

- ◯ SIM
- ◯ NÃO
- ◯ Parcialmente (justifique):

* 5. O mecanismo de navegação pelo documento foi útil para a atividade? Marque o nível de utilidade conforme:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

6. Marque a(s) receita(s) que foram úteis para a atividade.

- [ ] New - Nova Imagem
- [ ] Selecionar (clicar) Botão para Desenhar Figura Elipse
- [ ] Desenhar Figura Elipse
- [ ] Desenhar Figura Losango
- [ ] Desenhar uma Linha - Line Tool
- [ ] Colorir Figura
- [ ] Colorir Borda da Figura
- [ ] Copiar (Copy) Figura
- [ ] Colar (Paste) Figura
- [ ] Cortar (Cut)Figura
- [ ] Duplicar Figura (Duplicate Figure)
- [ ] Mover (Move) Figura
- [ ] Desfazer (Undo) Mover Figura
- [ ] Refazer (Redo) Mover Figura
- [ ] Conectar Figuras (Conecction Tool)
- [ ] Conectar Figuras 2 (Elbow Conecction Tool)
- [ ] Alinhar Figuras (Align Command)
- [ ] Alinhar Figuras ao Centro (Align Command Center)
- [ ] Trazer para frente (Bring to Front)
- [ ] Enviar para Trás (Send to Back)
- [ ] Agrupar Figuras (Group)
- [ ] Desagrupar Figuras (UnGrou
- [ ] Apagar Figura (Delete)
- [ ] Savar (save) Desenho
- [ ] NENHUMA DAS ANTERIORES

Justificativa (OPCIONAL):

7. Sobre os Hot-spots do framework sugeridos na(s) Receita(s), as informações sobre estes foram uteis para a realização da tarefa? Marque o nível de utilidade conforme:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

8. Marque a(s) Classes Hot-spots(s) da(s) receita(s) seguidas que foram úteis para a atividade.

- [ ] CH.ifa.draw.application.DrawApplication
- [ ] CH.ifa.draw.contrib.WindowMenu
- [ ] CH.ifa.draw.figures.AttributeFigure
- [ ] CH.ifa.draw.figures.ElbowConnection
- [ ] CH.ifa.draw.figures.LineConnection
- [ ] CH.ifa.draw.figures.LineFigure
- [ ] CH.ifa.draw.figures.RectangleFigure
- [ ] CH.ifa.draw.figures.TextFigure
- [ ] CH.ifa.draw.standard.CreationTool
- [ ] CH.ifa.draw.standard.DecoratorFigure
- [ ] CH.ifa.draw.standard.StandardDrawing
- [ ] CH.ifa.draw.standard.ToolButton
- [ ] CH.ifa.draw.util.CommandMenu
- [ ] CH.ifa.draw.util.UndoableCommand
- [ ] CH.ifa.draw.util.UndoableTool
- [ ] CH.ifa.draw.contrib.DiamondFigure.
- [ ] CH.ifa.draw.figures.EllipseFigure
- [ ] NENHUM DOS ANTERIORES

Justificativa (OPCIONAL):

9. Faltou alguma Classe Hot-spot importante para atividade na(s) Receita(s) seguida(s)?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais classes faltaram e por que (para que) as usou:

10. Na(s) Receita(s) seguida(s), faltou alguma informação importante sobre alguma Classe Hotspot para a atividade?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais informações faltaram e por que (para que) as usou:

11. Você redefiniu algum Método Hot-spot?

- ◯ NÃO
- ◯ SIM, liste o(s) métodos:

12. Faltou algum Método Hotspot importante para atividade na(s) Receita(s) seguida(s)?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais métodos faltaram e por que (para que) os usou:

13. Faltou alguma informação importante sobre algum Método Hotspot apresentado na(s) Receita(s) seguida(s)?

- ◯ NÃO SEI RESPONDER
- ◯ NÃO
- ◯ SIM

Se responder SIM, liste quais informações faltaram e por que (para que) as usou:

14. As informações sobre ASSINATURA das Classes Hot-spot foram úteis para o desenvolvimento da atividade? Marque o nível de utilidade:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

15. As informações sobre PADRÃO DE PROJETO foram úteis para o desenvolvimento da atividade? Marque o nível de utilidade:

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

16. As informações sobre COMENTÁRIO DE CÓDIGO das CLASSES Hot-spot foram úteis?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

17. As informações sobre MÉTODOS que podem ser REDEFINIDOS foram úteis para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

18. As informações sobre MÉTODOS que podem ser INVOCADOS foram úteis para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

19. As informações sobre COMENTÁRIOS SOBRE OS MÉTODOS foram úteis para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

20. A informação com EXEMPLO DE REDEFINIÇÃO do método hotspot foi útil para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

21. A informação com EXEMPLO DE INVOCAÇÃO do método hotspot foi útil para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

22. A informação com EXEMPLO DE INSTANCIAÇÃO DA CLASSE HOTSPOT foi útil para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

23. As informações sobre MÉTRICAS do framework JHotDraw foram úteis para o desenvolvimento da atividade?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

24. Esta atividade foi COMPLEXA de realizar?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justifique (OPCIONAL):

25. Você ficou SATISFEITO com o uso do Documento para realizar esta atividade?

Satisfação: Sensação agradável que sentimos quando as coisas correm à nossa vontade ou se cumprem a nosso contento.

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa:

26. Foi FÁCIL LOCALIZAR as informações necessárias para instanciação no Documento?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

27. As informações dispostas no documento estavam CLARAS, ou seja, facilmente compreensíveis e bem apresentadas?

- ◯ 2 – Concordo Fortemente
- ◯ 1 - Concordo
- ◯ 0 - Neutro - Não soube opinar
- ◯ -1 - Discordo
- ◯ -2 - Discordo Fortemente

Justificativa (OPCIONAL):

Anexo 5

Termo de Consentimento Livre e Esclarecido.

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

Você está sendo convidado (a) para participar da pesquisa intitulada INSTANCIAÇÃO DE FRAMEWORKS USANDO COOKBOOKS CONSTRUÍDOS A PARTIR ANÁLISE ESTÁTICA E ANÁLISE DINÂMICA DE CÓDIGO-FONTE. sob a responsabilidade dos pesquisadores Marcelo de Almeida Maia e Raquel Fialho de Queiroz Lafetá. Nesta pesquisa buscamos verificar os benefícios de um nova forma de documentação de frameworks, baseada no código fonte. Pretendemos verificar se o documento propostos são uma alternativa viável de documentação, isto é, que seja tão bom ou melhor que documentação gerada manualmente, em termos de taxa de acerto e tempo de execução da atividade. Para este fim, será necessário comparar o desempenho de desenvolvedores (participantes da pesquisa) utilizando as diferentes documentações.

O Termo de Consentimento Livre e Esclarecido será obtido pela pesquisadora Raquel Fialho e seu envio será realizada via web (e-mail). Na sua participação você irá realizar atividades reais de instanciação de frameworks que utilizam a linguagem JAVA, onde será necessária a compreensão da instanciação e modificações no código. Cada participante irá executar no mínimo duas atividades de instanciação. Estas atividade e as instruções de execução serão passadas pela pesquisadora após o aceite. O tempo de execução e código gerado será coletado e avaliado. Ao final da execução da atividade, você irá responder a um questionário sobre a atividade e o documento utilizado.

Os riscos destes experimentos consistem em: você poderá se sentir cansado com a atividade proposta, considerado semelhante ao risco de ficar desanimado com atividades que são habituais (do dia a dia) aos analistas. Poderá se sentir frustrado se não conseguir responder algum item do questionário. **Também, existe o risco de o participante ser identificado**. Porém, a pesquisa foi estruturada para que em nenhum momento você seja identificado. Os resultados da pesquisa serão publicados e ainda assim a sua identidade será preservada, visando não causar desconforto com relação aos resultados e respostas expostas. Apenas os pesquisadores envolvidos no estudo poderão avaliar as informações. A equipe responsável pelo estudo se compromete a manter em sigilo absoluto a identidade dos sujeitos participantes incluídos no estudo, inclusive na publicação dos resultados. A aplicação dos questionários propostos será conduzida de forma a não induzir, coibir, constranger ou reprimir qualquer comportamento ou informação relatada. Você não terá nenhum gasto e ganho financeiro por participar na pesquisa.

Os benefícios serão, ao final deste estudo, as informações geradas poderão trazer benefícios à você e outros analistas. Por meio das informações coletadas e analisadas serão gerados resultados compilados que podem auxiliar futuramente nas documentações de frameworks que são amplamente utilizados pelos analistas de TI em geral, atingindo diretamente ou indiretamente os sujeitos participantes. Além do mais, os participantes serão treinados e terão contato com este tipo de documentação, podendo contribuir para os seus conhecimentos sobre as técnicas que estão sendo desenvolvidos, tirando proveito em seu campo profissional.

Uberlândia, ... de .....de 20.....

_____ _____

(rubrica) Participante da pesquisa (rubrica) Assinatura dos pesquisadores

Todas as dúvidas referentes ao projeto de pesquisa serão esclarecidas. Os voluntários estarão livres para retirar o seu consentimento de participação no projeto sem nenhum prejuízo ou coação. Uma via original deste Termo de Consentimento Livre e Esclarecido ficará com você.

Qualquer dúvida a respeito da pesquisa, você poderá entrar em contato com a pesquisadora Raquel Fialho pelo contato: raquel.rafialho@gmail.com, skype: rafialho ou entrar em contato com a Faculdade de Computação – FACOM da Universidade Federal de Uberlândia. Campus Santa Mônica - Bloco 1A - Sala 1A236, Av. João Naves de Ávila, 2.121 - Bairro Santa Mônica, CEP 38400-902 - Uberlândia/MG. Poderá também entrar em contato com o Comitê de Ética na Pesquisa com Seres-Humanos – Universidade Federal de Uberlândia: Av. João Naves de Ávila, nº 2121, bloco A, sala 224, Campus Santa Mônica – Uberlândia –MG, CEP: 38408-100; fone: 34-32394131.

Uberlândia, ... de .....de 20.....

_____

Assinatura dos pesquisadores

Eu aceito participar do projeto citado acima, voluntariamente, após ter sido devidamente esclarecido.

_____

Participante da pesquisa

Anexo 6

Tabelas com Resultados dos Experimentos em Percentuais por Resposta.

Tabelas com Resultados – Experimentos UFU, FEMASS e Profissionais

	Grupo	Escala	Utilidade Navegação	Utilidade Informações	Facilidade de Localização	Clareza da Informação	Satisfação com Documentação	Complexidade da Atividade
UFU	Receitas	-2	14,29%	7,14%	21,43%	14,29%	14,29%	14,29%
		-1	0	7,14%	0	21,43%	14,29%	21,43%
		0	0	7,14%	0	14,29%	14,29%	7,14%
		1	35,71%	21,43%	50,00%	21,43%	21,43%	35,71%
		2	50,00%	57,14%	28,57%	28,57%	35,71%	21,43%
	Controle	-2	21,43%	14,29%	35,71%	14,29%	21,43	0
		-1	21,43%	21,43%	14,29%	7,14%	35,71	0
		0	0	21,43%	7,14%	28,57%	21,43	7,14%
		1	50,00%	28,57%	21,43%	35,71%	14,29	57,14%
		2	7,14%	14,29%	21,43%	14,29%	7,14	35,71%

Tabela 1A – Resultados Experimentos Alunos UFU, em percentuais por resposta.

	Grupo	Escala	Utilidade Navegação	Utilidade Informações	Facilidade de Localização	Clareza da Informação	Satisfação com Documentação	Complexidade da Atividade
FEMASS	Receitas	-2	6,67%	0	20,00%	6,67%	0	0
		-1	20,00%	13,33%	20,00%	33,33%	26,67%	13,33%
		0	20,00%	26,67%	6,67%	6,67%	26,67%	6,67%
		1	20,00%	53,33%	40,00%	33,33%	26,67%	33,33%
		2	33,33%	6,67%	13,33%	20,00%	20,00%	46,67%
	Controle	-2	26,67%	13,33%	33,33%	33,33%	20,00%	0
		-1	26,67%	20,00%	33,33%	26,67%	46,67%	6,67%
		0	13,33%	46,67%	20,00%	13,33%	13,33%	13,33%
		1	26,67%	20,00%	13,33%	26,67%	20,00%	20,00%
		2	6,67%	0	0	0,00%	0	60,00%

Tabela 2A – Resultados Experimentos Alunos FEMASS, em percentuais por resposta.

Tabelas com Resultados – Experimentos UFU, FEMASS e Profissionais

Profissionais	Grupo	Escala	Utilidade Navegação	Utilidade Informações	Facilidade de Localização	Clareza da Informação	Satisfação com Documentação	Complexidade da Atividade
	Receitas	-2	0	0	0	0	6,67%	6,67%
		-1	13,33%	26,67%	13,33%	26,67%	26,67%	40,00%
		0	26,67%	13,33%	0	6,67%	6,67%	13,33%
		1	53,33%	46,67%	60,00%	53,33%	46,67%	26,67%
		2	6,67%	13,33%	26,67%	13,33%	13,33%	13,33%
	Controle	-2	6,67%	6,67%	13,33%	6,67%	13,33%	0
		-1	46,67%	33,33%	20,00%	26,67%	53,33%	46,67%
		0	20,00%	60,00%	20,00%	20,00%	13,33%	0
		1	26,67%	0	46,67%	46,67%	20,00%	33,33%
		2	0	0	0	0	0	20,00%

Tabela 3A – Resultados Experimentos Profissionais, em percentuais por resposta.