Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells

Rafael Fontinele Ribeiro

Número de Ordem PPGCC: M001

Teresina-PI, Agosto de 2018

Rafael Fontinele Ribeiro

Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Pedro de Alcântara dos Santos Neto

Teresina-PI

Agosto de 2018

Rafael Fontinele Ribeiro

Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells/ Rafael Fontinele Ribeiro. – Teresina-PI, Agosto de 2018-

93 p. : il. (algumas color.) ; 30 cm.

Orientador: Pedro de Alcântara dos Santos Neto

Dissertação (Mestrado) – Universidade Federal do Piauí – UFPI
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação, Agosto de 2018.

1. Web. 2. Usabilidade. 3. *Usability smells*. 4. *Refactoring*. I. Pedro de Alcântara dos Santos Neto. II. Universidade Federal do Piauí. III. Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells.

CDU 02:141:005.7

Rafael Fontinele Ribeiro

Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Pedro de Alcântara dos Santos Neto
Orientador

Lincoln Souza Rocha
Convidado 1

Raimundo Santos Moura
Convidado 2

Teresina-PI
Agosto de 2018

Resumo

As aplicações Web têm passado por um grande e rápido crescimento nos últimos anos, tornando-se parte do nosso dia a dia. As grandes empresas utilizam essas aplicações para fornecer seus serviços. Por isso, é preciso garantir o desenvolvimento de aplicações de alta qualidade. Um dos atributos essenciais de uma aplicação Web, que determina diretamente seu sucesso ou fracasso, é a usabilidade. Esse atributo mede o quão adequada ao uso é a aplicação, tendo sido desenvolvidos diversos métodos para avaliá-lo. As avaliações de usabilidade têm como objetivo evidenciar os problemas de usabilidade da aplicação, facilitando a correção dos mesmos e, consequentemente, melhorando a usabilidade. O método de avaliação de usabilidade mais popular é o teste laboratorial, por conseguir avaliar diretamente os sentimentos dos usuários ao interagir com a aplicação final. Entretanto, a realização desse teste acarreta um alto custo e complexidade, devido necessidade de organização do ambiente laboratorial, convocação de participantes, avaliação de resultados subjetivos, etc. Assim, com o objetivo de simplificar a realização dessa avaliação e facilitar a descoberta de problemas de usabilidade, este trabalho apresenta uma abordagem para a detecção automática de indicativos de problemas, também conhecidos como *usability smells*. Essa abordagem é baseada na captura da interação do usuário em contexto de produção, ou seja, com o mesmo realizando suas atividades diárias livremente, e na análise automática dessa interação. De acordo com os padrões de ações analisados, a abordagem propõe a aplicação de algoritmos específicos para a detecção de *usability smells*, que os identificam e reportam informações importantes a respeito das detecções, visando facilitar a localização da origem do problema. Cada um dos *smells* propostos foi catalogado em termos de fundamentação, forma de detecção e *refactoring* sugerido. Um estudo realizado com a abordagem, demonstrou que os *smells* indicados por ela foram capazes de auxiliar na detecção e correção de problemas de usabilidade concretos em uma aplicação real. Apesar de não ser capaz de substituir completamente o teste laboratorial tradicional, realizado por avaliadores experientes, a abordagem proposta pode complementá-lo, auxiliando a detecção de problemas que são mais evidentes ao se analisar grandes conjuntos de dados, ou mesmo servir como uma alternativa em casos onde o custo financeiro e a demanda de tempo são restrições mais severas.

Palavras-chaves: Web. usabilidade. *usability smells*. *refactoring*.

Abstract

Web applications have gone through a large and rapid growth in recent years, becoming part of our day-to-day. Large companies use these applications to provide their services. Because of this, it is necessary to ensure the development of high quality applications. One of the essential attributes of a Web application, which directly determines its success or failure, is usability. This attribute measures how suitable for use is the application, and several methods have been developed to evaluate it. The usability evaluations aim to highlight the usability problems of the application, facilitating the correctness of them and, consequently, improving usability. The most popular usability evaluation method is the laboratory testing, because it can directly evaluate the users' feelings when interacting with the final application. However, performing this test entails a high cost and complexity, due to the necessity of organizing the laboratory environment, convening participants, evaluating subjective results, etc. Thus, in order to simplify the realization of this evaluation and facilitate the discovery of usability problems, this work presents an approach for the automatic detection of problem indicatives, also known as usability smells. This approach is based on the capture of user interaction in the context of production, that is, with the same one performing their daily activities freely, and in the automatic analysis of this interaction. According to the action patterns analyzed, the approach proposes the application of specific algorithms for the detection of usability smells, which identify and report important information about the detections, in order to facilitate the location of the origin of the problem. Each of the proposed smells was cataloged in terms of reasoning, form of detection and suggested refactoring. A study carried out with the approach demonstrated that the smells indicated by it were able to aid in the detection and correction of concrete usability problems in a real application. Although not able to completely replace traditional laboratory testing by experienced evaluators, the proposed approach can complement it by helping to detect problems that are most evident when analyzing large datasets, or even serve as an alternative in cases where the financial cost and the time demand are more severe constraints.

Keywords: Web. usability. usability smells. refactoring.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

AA	*Ações Alertas*
AC	*Ações Corretas*
AO	*Ações Obrigatórias*
AP	*Ações Problemáticas*
AST	*Abstract Syntax Tree*
AutoQUEST	*Automatic Quality Engineering of Event-driven Software*
BD	*Banco de Dados*
CSS	*Cascading Style Sheets*
EUSADUS	*Extensão da UseSkill com a Abordagem para Detecção de Usability Smells*
GDS	*Grupo das Demais Sessões*
GQM	*Goal, Question, Metric*
GSR	*Grupo de Sessões de Referência*
GUI	*Graphical User Interface*
ID	*Identificador*
IDE	*Integrated Development Environment*
IHC	*Interação Humano-Computador*
IU	*Interface de Usuário*
ISO	*International Organization for Standardization*
MUSE	*Mobile Usability Smell Evaluator*
QP	*Questão de Pesquisa*
RAM	*Random Access Memory*
SIGAA	*Sistema Integrado de Gestão de Atividades Acadêmicas*
TAE	*Transportable Applications Environment*

TI	*Tecnologia da Informação*
UEM	*Usability Evaluation Method*
UFPI	*Universidade Federal do Piauí*
UI	*User Interaction*
URL	*Uniform Resource Locator*
UsAGE	*User Action Graphing Effort*
USF	*Usability Smells Finder*
USOTF	*UseSkill OnTheFly*
WELFIT	*Web Event Logger and Flow Identification Tool*
XML	*Extensible Markup Language*

Lista de símbolos

$\cup$	União
$\sum$	Somatório
$\in$	Pertence
$\notin$	Não pertence

Sumário

Introdução

Contexto

Atualmente, a grande maioria das empresas utiliza a Tecnologia da Informação (TI) como meio para automatizar seus processos de negócio. Um elemento chave que determina o sucesso de uma implementação de tecnologia aliada a um negócio são os seus usuários. Contudo, os usuários são constantemente ignorados na tomada de decisões a respeito de tecnologias, devido a restrições como indisponibilidade de tempo, falta de procedimentos padronizados ou presença de outras prioridades (PAUL; YADAMSUREN; ERDELEZ, 2012).

O sucesso de um sistema de informação é altamente dependente da usabilidade de sua interface. A interface de uma aplicação deve permitir ao usuário ter uma boa consciência sobre o estado do sistema e, durante a execução, trazer facilidades que ajudem a reduzir o número de passos necessários para transformar uma intenção em uma ação (CASSINO; TUCCI, 2011). Assim, com o objetivo auxiliar na melhoria da usabilidade de interfaces de aplicações Web e, consequentemente, na melhora da experiência do usuário, vários métodos de avaliação de usabilidade foram desenvolvidos (FERNANDEZ; INSFRAN; ABRAHÃO, 2011).

A definição de usabilidade mais intuitiva pode ser dada como a propriedade do sistema que define seu grau de simplicidade em termos de aprendizado, memorização e eficiência. Há bastante tempo, o valor de um sistema de software tem sido medido com base em sua interface gráfica e o poder de expressão e comunicação associado (CASSINO et al., 2015). Um sistema com boa usabilidade permite aos usuários alcançarem seus objetivos com facilidade, rapidez e eficiência, ao mesmo tempo em que gera satisfação com os resultados (JAAFAR et al., 2008)

A avaliação de usabilidade, por sua vez, consiste na aplicação de metodologias para tentar mensurar a usabilidade do sistema e detectar os problemas específicos que estão afetando negativamente a sua interface. Uma ampla variedade de técnicas de avaliação de usabilidade tem sido proposta. Cada técnica tem seus próprios requisitos e, geralmente, diferentes técnicas explicitam diferentes problemas de usabilidade (IVORY; HEARST, 2001).

Escopo

As aplicações Web auxiliam em muitas das atividades cotidianas, como compras, leitura de notícias, interação social, realização de transações bancárias, planejamento de viagens, marcação de consultas, dentre outros. Todos os dias, novas aplicações Web são desenvolvidas, ampliando as possibilidades para realizar tarefas confortavelmente em casa (GRIGERA et al., 2017).

Uma das razões para contínuo e impressionante crescimento da Web na última década é a disponibilidade de ferramentas e facilidades para a criação e publicação de informações. Mais que simplesmente facilitar a criação de páginas, essas ferramentas transformam a construção de uma aplicação Web completamente integrada em uma tarefa simples (VASCONCELOS; BALDOCHI JR., 2012). Como resultado, a Web se tornou um ambiente comum para aplicações computacionais. Isso despertou um interesse crescente sobre a usabilidade dessas aplicações entre pesquisadores e desenvolvedores (VARGAS; WEFFERS; ROCHA, 2010).

Muitas vezes os usuários acabam frustrados por serem incapazes de achar o que procuram (SCHOLTZ, 2001) e, se o sistema é difícil de usar, terminam por ignorá-lo e negligenciá-lo (JAAFAR et al., 2008). Já é bem conhecido que há vários benefícios associados com o melhoramento da usabilidade de aplicações Web. A usabilidade tem impacto nas taxas de conversão da aplicação, eficiência de trabalho dos usuários e na imagem de uma empresa. Em adição, o custo de suporte do usuário pode ser reduzido aumentando-se a usabilidade (KIURA; OHIRA; MATSUMOTO, 2009).

Os métodos de teste de usabilidade se destacam como a classe de métodos de avaliação de usabilidade mais populares para a Web (FERNANDEZ; INSFRAN; ABRAHÃO, 2011), sendo os únicos capazes de avaliarem a interação dos usuários com o sistema real. Essa classe inclui os métodos em que um avaliador observa a interação do usuário com a interface e tenta encontrar problemas de usabilidade baseado em seu comportamento e experiências durante a avaliação.

Motivação

Dentre os métodos de teste de usabilidade, o teste laboratorial é um dos mais amplamente aceitos (TULLIS et al., 2002). Apesar de ser um método eficiente para a detecção de problemas de usabilidade, há um alto custo e complexidade associados à busca e movimentação de participantes, preparação da infraestrutura, condução do teste e à coleta e análise de resultados (VARGAS; WEFFERS; ROCHA, 2010).

O teste laboratorial é sempre considerado como um processo dispendioso e exaustivo, mas está se tornando mais importante, já que muitas de nossas operações comerciais e

atividades sociais são processadas e concluídas diretamente na Internet (LI; KIT, 2005). Assim, tem havido muitos trabalhos relacionados ao melhoramento do desempenho do mesmo (AHMAD; SULAIMAN; JOHARI, 2010).

Apesar de diversas propostas serem bem sucedidas na tarefa de reduzir as dificuldades relacionadas ao teste (FERNANDEZ; INSFRAN; ABRAHÃO, 2011), muitas delas ainda sofrem com problemas próprios como a necessidade de organização do ambiente laboratorial e alocação de participantes e especialistas para o teste, resultados pouco representativos do contexto de uso real, configuração e/ou organização complexa, etc.

Declaração do Problema

A condução do teste laboratorial deveria ser simples e ter bom custo-benefício, para que pudesse ser mais amplamente adotado e, consequentemente, causasse uma melhora geral na usabilidade das aplicações Web. A necessidade de um mecanismo para a execução do teste laboratorial de forma eficiente é especialmente importante no caso de pequenas empresas, onde o custo do mesmo é um fator mais impactante. Contudo, a forma de execução tradicional do teste laboratorial é complexa, demorada e custosa. Ao possuir tais dificuldades, sua introdução no processo de desenvolvimento é desestimulada e, como resultado, o aspecto de usabilidade, que é de grande importância para a satisfação do cliente, acaba por ser ignorado.

A falta de usabilidade de uma aplicação Web desencoraja o seu uso, diminuindo a convergência dos usuários. Além disso, ela pode se tornar confusa e difícil de manter para os desenvolvedores. Em adição, a imagem da empresa associada a essa aplicação pode ser prejudicada, fora o prejuízo financeiro causado pela perda de clientes. Clientes insatisfeitos também tendem a compartilhar suas insatisfações com amigos, o que afasta potenciais novos clientes e torna a situação ainda mais crítica (TRAVIS, 2007).

Uma das possibilidades para melhorar a usabilidade de uma aplicação é a utilização da técnica de *refactoring* (OPDYKE, 1992), que foi trazida para o campo da usabilidade de aplicações Web (GRIGERA; GARRIDO; RIVERO, 2014; GARRIDO; ROSSI; DISTANTE, 2011). *Refactoring* é o processo de melhorar a estrutura interna de um código, de forma a não modificar o seu comportamento (FOWLER; BECK, 1999). O poder da refatoração está em ajudar os não especialistas a identificar potenciais problemas e boas soluções passo a passo para esses problemas (GRIGERA et al., 2017).

Os *refactorings* são comumente utilizados para a eliminação de *code smells*. Um *code smell* indica um sinal no código ou na aplicação que, mesmo não sendo um erro, torna difícil a sua compreensão, manutenção e evolução (ALMEIDA, 2014). Adaptando-se essas ideias para o contexto de usabilidade, criou-se o conceito de *usability smells* (GARRIDO; ROSSI; DISTANTE, 2011), que segue a mesma definição de *code smells*, porém aplicada a

interfaces em vez de código fonte.

Os *usability smells* podem ser utilizados para reduzir as dificuldades do teste laboratorial, uma vez que proveem um guia direto para encontrar os principais comportamentos associados à problemas de usabilidade. Entretanto, procurar por *usability smells* manualmente pode ser uma tarefa bem laboriosa, principalmente quando realizada em aplicações grandes, que geralmente possuem dezenas (ou até centenas) de interfaces diferentes. Além disso, essa técnica ainda é pouco explorada no contexto das aplicações Web e o número de ferramentas que objetivam auxiliá-la computacionalmente é bem reduzido.

Indicar *usability smells* automaticamente pode reduzir os esforço necessário na identificação de problemas de usabilidade, tradicionalmente encontrado no teste laboratorial, uma vez que esses *smells* guiam o avaliador nessa tarefa. Consequentemente, o nível de experiência exigido do mesmo diminui, o que, por sua vez, pode incentivar a adoção do teste laboratorial no desenvolvimento de aplicações Web. E com essa adoção, podem-se produzir aplicações com maior nível de usabilidade.

Objetivos

O objetivo geral deste trabalho é apresentar uma abordagem para automatizar a detecção de *usability smells* em aplicações Web. A abordagem se propõe a reduzir o custo e complexidade associados à realização do teste laboratorial, permitindo que o mesmo possa ser mais facilmente executado. Os objetivos específicos deste trabalho são:

- Criar uma abordagem simples e prática, que possa ser aplicada mesmo por avaliadores com baixo nível de experiência em usabilidade e com pouco esforço;
- Apresentar uma forma de detecção automática para *usability smells* existentes na literatura, de acordo com as definições da abordagem;
- Apresentar novos *usability smells* não encontrados na literatura, bem como sua forma de detecção automática;
- Criar uma ferramenta que implemente a abordagem proposta, de modo a permitir a sua utilização em larga escala, com baixo custo de aplicação e manutenção;

Embora a automação completa da detecção de problemas de usabilidade seja um desejo desta pesquisa, sabe-se que, devido à natureza subjetiva dessa tarefa, isso é algo muito difícil de alcançar. Portanto, o intuito deste trabalho é gerar avanços que permitam aproximar-se ao máximo desse cenário ideal, facilitando a detecção de *usability smells*, que são indicativo de possíveis problemas.

Estrutura da Dissertação

Esta dissertação está dividida em quatro capítulos numerados, mais a Introdução. A Introdução mostra o contexto da área explorada, a motivação para atuar na mesma, a declaração do problema atacado, os objetivos que pretende-se alcançar com o desenvolvimento do trabalho e um breve resumo sobre o conteúdo descrito em cada capítulo.

O Capítulo 1 descreve os conceitos que são fundamentais para um bom entendimento do trabalho. Inicialmente é apresentada a definição de usabilidade e as formas de avaliação existentes de acordo com a taxonomia proposta por Ivory e Hearst (2001). Em seguida, é discorrido sobre as vantagens e as principais formas de automação de avaliações de usabilidade. Por fim, são abordados a definição, origem e aplicações dos *usability smells*.

O Capítulo 2 apresenta os principais trabalhos que descrevem abordagens semelhantes à proposta nesta dissertação. Alguns dos principais trabalhos relacionados à detecção de problemas de usabilidade são comentados brevemente, mas a prioridade foi dada àqueles que buscam a identificação de *usability smells*, visto que esse é o principal conceito explorado nesta dissertação.

O Capítulo 3 apresenta o funcionamento da abordagem proposta. Os procedimentos envolvidos em cada uma de suas etapas são detalhados, com foco nas estratégias de detecção utilizadas e na ideia por trás de cada um dos *smells* incorporados, assim como a forma de implementação computacional desses procedimentos. Também discorre-se sobre as limitações da abordagem e como tentou-se amenizar cada uma delas.

Por fim, o Capítulo 4 apresenta os resultados das avaliações realizadas com a abordagem proposta. Após estabelecido o contexto das avaliações, são discutidos seus pontos positivos e negativos, bem como os achados referentes à detecção realizada para cada *usability smell* e à implementação da abordagem como um todo.

Ao fim da estrutura de capítulos numerados, também são apresentadas as Considerações Finais. Além de recapitular, de forma sintetizada, todos os pontos do trabalho, essa parte também discorre sobre os aspectos que precisam ser tratados durante a continuidade da pesquisa, de acordo com as conclusões inferidas a partir da realização desta pesquisa.

1 Fundamentação Teórica

Este capítulo apresenta em detalhes os principais conceitos utilizados no trabalho. Primeiramente apresenta-se a definição de usabilidade e os principais métodos de avaliação da mesma, focando no teste laboratorial, que será o método abordado pela proposta deste trabalho. Em seguida, serão mostrados a origem e as definições relacionadas à *usability smells* e *refactorings* de usabilidade, visto que são temas relativamente recentes e necessitam de uma maior fundamentação.

1.1 Usabilidade

A definição de usabilidade mais citada nos trabalhos lidos durante o levantamento bibliográfico realizado para esta pesquisa, e mais aceita na área de Interação Humano-Computador (IHC), é a da norma ISO 9241-11, a saber: "é o âmbito no qual um produto pode ser utilizado por usuários específicos para alcançar objetivos específicos com eficácia, eficiência e satisfação em um contexto específico de uso" (FERNANDEZ; INSFRAN; ABRAHÃO, 2011; ISO, 1998).

Mais particularmente em relação ao contexto das aplicações Web, utilizado nesta dissertação, a usabilidade também pode ser definida como o âmbito no qual um sistema computacional permite aos seus usuários, em um dado contexto de uso, alcançar objetivos específicos efetivamente e eficientemente, enquanto promovendo sentimentos de satisfação (IVORY; HEARST, 2001). Nessa visão, usabilidade implica na interação dos usuários com um produto de software e pode ser vista como a capacidade de atender às expectativas do consumidor (FERNANDEZ; INSFRAN; ABRAHÃO, 2011).

Um **problema de usabilidade**, por sua vez, é um problema que ocorre durante o uso de um produto e reduz um ou mais dos fatores eficácia, eficiência e satisfação para um contexto específico de uso. Nas aplicações Web, esses problemas podem ser causados pelo *design*, a funcionalidade, a performance, e outros aspectos (HARMS; GRABOWSKI, 2014).

1.2 Métodos de Avaliação de Usabilidade

Um Método de Avaliação de Usabilidade (*Usability Evaluation Method* ou UEM) é um procedimento composto de uma série de atividades bem definidas para coletar dados de uso relacionados à interação do usuário final com um produto de software e/ou como as propriedades específicas desse produto de software contribuem para alcançar um certo nível

de usabilidade. Os UEMs foram anteriormente desenvolvidos especificamente para avaliar interfaces WIMP (*Window*, *Icon*, *Menu*, *Pointing device*), contudo, com o crescimento em importância das aplicações Web, novos e adaptados UEMs têm surgido para abordar esse tipo de interface de usuário (IU) (FERNANDEZ; INSFRAN; ABRAHÃO, 2011).

Avaliar uma aplicação Web, em particular, consiste em verificar se o *design* da aplicação permite aos usuários recuperar informações e navegar através dos conteúdos facilmente, e invocar os serviços e operações disponíveis. Isso, portanto, implica não apenas em ter os conteúdos e serviços apropriados disponíveis na aplicação, mas também em torná-los de fácil acesso para os usuários (MATERA; RIZZO; CARUGHI, 2006). A avaliação de usabilidade é uma parte importante do processo de *design* de interfaces e implica em muitas atividades, as mais comuns sendo (IVORY; HEARST, 2001):

- **Captura**: coletar dados de usabilidade, como tempo de finalização da tarefa, erros, violação de *guidelines*, e avaliações subjetivas;
- **Análise**: interpretar dados de usabilidade para identificar problemas de usabilidade na interface;
- **Crítica**: sugerir soluções ou melhorias para mitigar os problemas.

Com o intuito de facilitar a discussão sobre as avaliações de usabilidade, Ivory e Hearst (2001) propuseram uma taxonomia para agrupar os UEMs em relação a diferentes aspectos que serão discutidos ao longo deste capítulo. O "Tipo de Esforço"[1] é um desses aspectos, e descreve o esforço humano requerido para a execução do método de avaliação. Os tipos possíveis são:

- **Esforço Mínimo**: não requer o uso de interface ou modelagem;
- **Desenvolvimento de Modelo**: requer que o avaliador desenvolva um modelo de IU e/ou de usuário para poder aplicar o método;
- **Uso Informal**: requer a execução de tarefas livremente escolhidas, isto é, uso irrestrito do sistema por um usuário ou avaliador;
- **Uso Formal**: requer a execução de tarefas especialmente selecionadas, isto é, uso restrito do sistema por um usuário ou avaliador.

Os papéis envolvidos em uma avaliação de usabilidade tipicamente incluem um desenvolvedor (que implementa a aplicação e/ou IU), um avaliador (que planeja e conduz

[1] O termo original utilizado no trabalho de referência é "Nível de Esforço". Contudo, como a ideia de nível pressupõe uma hierarquia, e não é bem claro qual das técnicas envolvidas requer mais esforço, o termo foi alterado nesta dissertação para aumentar a coerência.

as sessões de avaliação) e um usuário ou sujeito (que participa nas sessões de avaliação) (BOWMAN; GABBARD; HIX, 2002). Já quanto ao "Tipo de Avaliação", segundo a taxonomia proposta por Ivory e Hearst (2001), os UEMs podem ser divididos em cinco classes:

- **Teste**: um avaliador observa os usuários interagindo com uma interface para determinar problemas de usabilidade. Esses usuários geralmente são bem representativos do público alvo, e são reunidos em um ambiente artificial e solicitados a completar algumas poucas, mas realísticas, tarefas predefinidas (HONG et al., 2001). O benefício do teste com usuários sobre os outros métodos de avaliação é que ele captura dados de uso real e as experiências dos usuários. O lado negativo, entretanto, é que ele requer o recrutamento de usuários e o gasto de tempo e recursos para os avaliadores desenharem os testes, analisarem os resultados, descobrirem os problemas e encontrarem soluções para esses problemas (GRIGERA et al., 2017). O protocolo Pensar em Voz Alta[2] (*Think Aloud*) e o teste laboratorial tradicional e remoto são alguns exemplos de métodos de teste bem populares;

- **Inspeção**: um avaliador usa uma série de critérios ou heurísticas para identificar potenciais problemas de usabilidade em uma interface. A principal vantagem dos métodos de inspeção é que eles são altamente rentáveis, uma vez que não exigem custo nem esforço com o recrutamento de usuários. Além disso, eles podem ser utilizados mais cedo no ciclo de vida da aplicação, já que não exigem que o sistema esteja implementado ainda, podendo ser aplicados em protótipos ou até mesmo na simples especificação da interface. Dentre os principais métodos de inspeção estão a Avaliação Heurística e o Passo a Passo Cognitivo (NIELSEN, 1994);

- **Investigação**: os usuários proveem *feedback* sobre uma interface através de alguma técnica de investigação. Esse método é capaz de reunir entradas subjetivas dos participantes, como as suas preferências e sentimentos. Porém, como são focados apenas em coletar dados subjetivos dos usuários, a maioria deles é usado em combinação com outros tipos de métodos, para realizar uma avaliação mais completa. Os métodos de investigação mais representativos são questionários, entrevistas e o Grupo Focal (FERNANDEZ; INSFRAN; ABRAHÃO, 2011);

- **Modelagem Analítica**: um avaliador emprega modelos formais para analisar o sistema com o objetivo de identificar problemas e gerar predições de usabilidade (MOSQUEIRA-REY et al., 2009). Esses modelos visam substituir o papel do usuário durante a avaliação e permitem predizer seus objetivos, tempo de aprendizado e complexidade do *design* do sistema, entre outras informações. Apesar de

[2] Consiste em analisar a resposta do usuário enquanto o mesmo verbaliza seus pensamentos ao executar as tarefas (SOMEREN et al., 1994).

possuírem baixo custo, são pouco explorados no domínio Web devido à dificuldade e complexidade de se criar modelos bem representativos. O modelo GOMS (*Goals*, *Operators*, *Methods*, *Selection rules*) é um dos mais utilizados;

- **Simulação**: um avaliador emprega modelos para imitar o usuário interagindo com uma interface, através de algum algoritmo de simulação ou da análise de dados de uso (e.g. modelos de Redes de Petri, *Information Scent*), e reportar os resultados dessa interação. Uma das grandes vantagens desse método é que possibilita simular vários cenários e *designs* diferentes, apenas alterando-se alguns parâmetros. Contudo, a ausência de usuários reais produz um contexto e resultados artificiais. Das 5 classes de métodos essa é a menos utilizada. Além disso, poucos métodos de avaliação podem ser considerados como apenas simulação, já que apresentam características de outros tipos de métodos, especialmente da Modelagem Analítica. É possível simular atividades, erros, e diversas outras medidas quantitativas (FERNANDEZ; INSFRAN; ABRAHÃO, 2011).

A abordagem proposta nesta dissertação pode ser categorizada na classe de Teste, mais especificamente, como uma forma alternativa de execução do teste laboratorial[3]. Esse método de avaliação foi escolhido como foco deste trabalho por ser um dos mais bem aceitos e completos, conseguindo avaliar tanto métricas quantitativas, como eficácia e eficiência, quanto qualitativas, como a satisfação (FERNANDEZ; INSFRAN; ABRAHÃO, 2011).

Entretanto, o teste tradicional[4] consome muito tempo para ser executado com grandes quantidades de participantes, uma vez que exige uma quantidade de esforço considerável para organizá-los, observá-los e analisar os resultados. Consequentemente, os dados desse tipo de avaliação tendem a refletir apenas poucas pessoas e são em maior parte qualitativos. Esse número pequeno torna difícil cobrir todas as tarefas possíveis em uma aplicação (HONG et al., 2001). Além disso, um pequeno conjunto de participantes pode não ser adequado para se encontrar a maior parte dos problemas (SPOOL; SCHROEDER, 2001).

Uma alternativa ao teste tradicional são os testes remotos, que não exigem o deslocamento dos usuários até o laboratório, nem que os mesmos estejam presentes no mesmo local que o avaliador. Esses testes também podem ser assíncronos, ou seja, não necessitam que os usuários sejam acompanhados pelos especialistas em tempo real,

[3] Apesar de ser baseada no teste laboratorial, a abordagem proposta em si não pode ser considerada como um tipo de teste laboratorial, uma vez que não necessita de um ambiente artificial para ser utilizada, como será melhor detalhado nas próximas seções.

[4] Os termos “teste laboratorial” e “teste tradicional’ são usados para denominar o mesmo tipo de avaliação nesta dissertação, com o termo “tradicional” sendo usado apenas para enfatizar que o mesmo não utiliza nenhum tipo de auxílio de ferramentas de automação.

podendo a análise dos resultados ser realizada a qualquer momento após a finalização do teste (BRUUN et al., 2009).

Apesar de reduzirem algumas das dificuldades envolvidas no teste tradicional, os testes remotos ainda exigem a presença de especialistas e o recrutamento de usuários, acarretando em grandes custos e esforço de organização e condução. Como uma forma de facilitar ainda mais a execução do teste remoto, surgiram os testes automatizados, que podem substituir total ou parcialmente o papel do avaliador especialista, auxiliando em todas as etapas do teste. Essa automação será discutida em mais detalhes na próxima subseção.

A abordagem proposta pode ser utilizada de forma remota e assíncrona, e contempla a avaliação de diversas métricas quantitativas a respeito do sistema. Contudo, como ela não envolve a participação direta do usuário, não é possível obter *feedback* do mesmo e, consequentemente, as métricas qualitativas não podem ser mensuradas.

1.3 Automação dos Métodos de Avaliação de Usabilidade

Diversas abordagens foram desenvolvidas para automatizar aspectos das avaliação de usabilidade, com o objetivo de aumentar a cobertura e sistematicidade dos resultados. A automação das avaliações de usabilidade tem muitas vantagens potenciais sobre a avaliação não-automatizada, como (IVORY; HEARST, 2001):

- Redução do custo da avaliação;
- Incremento da consistência dos erros cobertos;
- Predição dos custos de tempo e erro através de todo o *design*;
- Redução da necessidade de experiência em avaliações entre os avaliadores individuais;
- Incremento da cobertura das funcionalidades avaliadas;
- Possibilidade de comparação entre *designs* alternativos;
- Incorporação da avaliação na fase de *design*, etc.

Utilizar uma abordagem automatizada para avaliar aplicações Web pode, não apenas ajudar a poupar tempo e dinheiro com a organização da avaliação, mas também melhorar a consistência e qualidade do *design* da interface, e aprimorar a aplicação sistemática de padrões de usabilidade (BRINCK; HOFER, 2002). A taxonomia proposta por Ivory e Hearst (2001) também classifica os métodos de avaliação de acordo com qual

aspecto da avaliação de usabilidade é automatizado, ou seja, o seu "Tipo de Automação"[5], que pode ser:

- **Nenhum**: não há nível de automação suportado, ou seja, o avaliador executa todos os aspectos da avaliação manualmente;
- **Captura**: grava dados de usabilidade automaticamente (e.g., registro do uso da interface em *log*);
- **Análise**: identifica problemas de usabilidade ou gera informações que facilitam a sua identificação automaticamente; e
- **Crítica**: analisa e sugere correções para os problemas encontrados automaticamente.

A automação tem sido usada predominantemente de duas formas no teste de usabilidade: captura automática de dados de uso e análise automática desses dados de acordo com algumas métricas ou um modelo (IVORY; HEARST, 2001). A ideia de reunir métricas de usabilidade a partir de eventos de interação do usuário tem sido considerada há muito tempo como uma valiosa fonte de informação, mesmo fora do domínio Web (GRIGERA et al., 2017).

Diversas abordagens capturam os eventos de interação do usuário e realizam algum tipo de análise de *logs* para ajudar os especialistas na descoberta de padrões. Os resultados geralmente são apresentados com ferramentas de visualização sofisticadas que permitem comparar sequências de eventos de usuário com uma sequência ótima. Contudo, essas ferramentas raramente proveem sugestões para ajudar os desenvolvedores a melhorarem seus artefatos. Nesse caso, o especialista ainda é necessário para detectar problemas de usabilidade concretos nos desvios entre as sequências de eventos, e a solução para esses problemas. Além disso, o conjunto de problemas de usabilidade que podem ser reconhecidos comparando sequências de eventos é limitado (GRIGERA et al., 2017).

Os *logs* usados pelas abordagens de automação podem ser capturados no lado do servidor ou no lado do cliente. A captura no lado do servidor é tecnicamente mais simples, mas os dados capturados revelam apenas informações relacionadas às páginas que os usuários visitaram. Em contraste, a captura de dados no lado do cliente é computacionalmente mais complexa, mas os dados capturados revelam informações mais detalhadas (i.e., ações e os elementos da IU onde elas ocorreram) (SANTANA; BARANAUSKAS, 2015).

Apesar das abordagens de automação reduzirem o trabalho do especialista e facilitarem o recrutamento de usuários, ainda é necessário que os mesmos reservem tempo exclusivamente para participarem do teste. Além disso, a usabilidade é influenciada pelo

[5] Também pode ser referido como "Etapa de Automação", uma vez que cada item corresponde a uma das etapas geralmente presentes em uma avaliação de usabilidade.

contexto de uso (HARMS; GRABOWSKI, 2014), e os usuários tendem a agir de forma diferente do habitual durante o teste laboratorial, devido ao ambiente artificial e do fato de estarem sendo monitorados, dificultando a obtenção de resultados representativos do contexto real (CASTILLO; HARTSON; HIX, 1998).

Uma forma de aproximar os resultados do contexto real é a realização do teste em ambiente não controlado. O teste em ambiente não controlado requer diferentes técnicas de análise, já que não há eventos ótimos com os quais se comparar, e a quantidade de dados é potencialmente grande e desorganizada. Apesar de tudo, essa abordagem tem ao menos dois benefícios sobre os métodos controlados: primeiro, elas são menos custosas, já que não requerem o recrutamento de voluntários e a criação de tarefas, e segundo, elas podem encontrar outros problemas que só aparecem em um ambiente real e não controlado, onde os usuários não estão restritos a um conjunto limitado de tarefas (GRIGERA et al., 2017).

A abordagem proposta nesta dissertação utiliza a captura de *logs* no lado do cliente e em ambiente não controlado, como uma forma de obter resultados tão significativos quanto possível. Além disso, ela provê relatórios detalhados sobre os *logs* analisados e dá sugestões para o avaliador, com o objetivo de reduzir (ou eliminar) a necessidade de um especialista. Assim, ela cobre a Captura, Análise e Crítica automáticas.

1.4 *Usability Smells*

Um *code smell* é algo no código ou na aplicação que, mesmo não sendo um erro, torna difícil a sua compreensão, manutenção e evolução. Tipicamente, esses *code smells* mostram a necessidade de o programador alterar algo no seu código, ou seja, aplicar um *refactoring* na estrutura do seu programa (ALMEIDA, 2014).

Os *code smells* são mais conhecidos pelo trabalho de Fowler e Beck (1999), onde apresenta-se uma catálogo de *code smells* bem definidos para ajudar os desenvolvedores a detectarem e corrigirem esses problemas em seus códigos. Nesse mesmo trabalho ele também define o conceito de *refactoring* como sendo "o processo de alterar um sistema de software de tal maneira que o comportamento externo do código não seja alterado, contudo melhorando a sua estrutura interna".

A ideia dos *usability smells* surgiu ao estudar-se os *code smells* de Fowler. Tanto no código como na interface, os erros e anomalias são comuns e obrigam a uma manutenção constante. O desenvolvedor é assim obrigado a aumentar o tempo gasto na manutenção se tiver como objetivo tentar manter a legibilidade da aplicação. Assim, questionou-se se esses *smells* não poderiam existir também na interface gráfica das aplicações (ALMEIDA, 2014).

Por ser um conceito relativamente recente, cada trabalho utiliza uma definição

levemente diferente para *usability smell*. Harms e Grabowski (2014), por exemplo, definem um *usability smell* como "um comportamento excepcional do usuário que pode indicar problemas de usabilidade".

Neste trabalho, será utilizada a definição de *usability smells* como sendo "indicadores de possíveis problemas que precisam de *refactorings*, onde os problemas estão relacionados a algum aspecto de qualidade no uso da aplicação Web: eficácia, eficiência ou satisfação no uso" (GARRIDO; ROSSI; DISTANTE, 2011 apud GRIGERA et al., 2017). Essa definição foi escolhida por ser a encontrada no trabalho mais completo relacionado à detecção de *usability smells* e a mais relacionada aos objetivos deste trabalho. Apesar das pequenas diferenças em definição, todos os trabalhos encontrados concordam no fato de que *usability smells* são indicativos de problemas de usabilidade.

Os *usability smells*, assim como os *code smells*, não são considerados erros, mas sim anomalias na aplicação que tornam difícil a sua compreensão, manutenção e evolução. O aparecimento desse tipo de *smells* indica ao programador que a aplicação necessita de alterações. As aplicações não estão erradas, mas o usuário percebe que as tarefas que tenta executar não têm a performance desejada (ALMEIDA, 2014).

Assim como ocorre com os *code smells*, a cada *usability smell* podemos associar um ou mais *refactorings*. Os *refactorings* de usabilidade podem ser definidos como mudanças na navegação, apresentação ou regras de negócio da aplicação com o propósito de melhorar a sua usabilidade, enquanto preserva a funcionalidade esperada e resultado (GARRIDO; ROSSI; DISTANTE, 2011 apud GRIGERA et al., 2017).

Detectar *usability smells* pode ser uma tarefa bem exigente que requer tempo e experiência. Contudo, essa tarefa pode ser bem melhorada com automação em diferentes níveis. Isso ajuda a tornar essa abordagem prática e útil para todos os desenvolvedores, independente da sua experiência com usabilidade. Além disso, há muitas fontes de *guidelines* de usabilidade e boas práticas na literatura que podem ser transformadas em *usability smells* (GRIGERA et al., 2017).

A abordagem apresentada nesta dissertação propõe estratégias para a detecção automática de diferentes *usability smells*. Esses *smells* foram, em parte, retirados de outros trabalhos já publicados, sendo que, para alguns desses, ainda não havia estratégia de detecção automática. A outra parte foi definida e catalogada neste trabalho, baseada nas boas práticas e na experiência do autor com avaliações de usabilidade. A cada *smell* catalogado também foi associado um *refactoring*, com o objetivo de ajudar a identificar e corrigir os possíveis problemas relacionados a ele.

1.5 Considerações Finais

Este capítulo apresentou os principais conceitos utilizados nesta dissertação. Foram mostradas as definições de usabilidade e *usability smells*, bem como as principais formas de detecção de problemas de usabilidade, de acordo com a taxonomia apresentada. Os motivos que levaram à escolha de cada elemento da abordagem proposta também foram mais detalhados. O próximo capítulo continua a apresentar mais informações reunidas nesta pesquisa, mostrando os trabalhos já existentes com propostas similares à apresentada nesta dissertação.

2 Trabalhos Relacionados

Este capítulo apresenta alguns dos trabalhos desenvolvidos com o objetivo de facilitar a detecção de problemas de usabilidade, de maneira similar à proposta desta dissertação. Será comentado, de maneira geral, sobre vários trabalhos que proveem algum tipo de auxílio ao teste laboratorial. Além disso, serão apresentados, de maneira mais detalhada, os trabalhos cujo foco é a automação da detecção de *usability smells*, e que ajudaram a fundamentar este trabalho teoricamente e descobrir métodos e técnicas usados para a implementação. O procedimento de pesquisa que levou à identificação de todos esses trabalhos também será explicitado.

2.1 Procedimento de Pesquisa

Inicialmente, o autor deste trabalho participou da extensão de um mapeamento sistemático, ainda em progresso, sobre "ferramentas que apoiam de forma automática a realização de avaliações de usabilidade em aplicações Web", cujos resultado podem ser parcialmente vistos em Souza (2016). Esse mapeamento ajudou a identificar vários trabalhos com foco em reduzir as dificuldades encontradas no teste laboratorial. Em especial, o trabalho de Grigera, Garrido e Rivero (2014) foi o que motivou o início desta pesquisa.

Tendo em conta que havia poucos trabalhos que abordavam o tema da detecção automática de *usability smells* entre os trabalhos identificados pelo mapeamento realizado, percebeu-se a necessidade de conduzir uma nova pesquisa. Assim, inicialmente tentou-se conduzir uma pesquisa informal pelo termo "usability smells" na base de busca Google Scholar, que indexa trabalhos das principais bases existentes. A partir dessa pesquisa, percebeu-se que a quantidade de trabalhos relacionados à esse tema era bem reduzida, o que não justificaria a condução de um novo mapeamento. Assim, optou-se por realizar apenas uma "pesquisa sistemática" sobre o tema.

O trabalho de Grigera et al. (2017) foi identificado durante a pesquisa informal realizada. Esse é um dos trabalhos mais recentes e completos sobre *usability smells*, motivo pelo qual ele foi escolhido como referência. A partir desse trabalho foram definidas as palavras-chave para a condução da pesquisa sistemática. Essas palavras-chave foram usadas para montar a *string* de busca utilizada, sendo ela:

usability AND (smell OR smells OR refactoring OR refactorings)

Foi escolhida uma *string* bem abrangente para a pesquisa, pois restringi-la muito

poderia deixar de fora alguns dos já poucos trabalhos existentes. As bases de busca escolhidas foram Scopus e Engineering Village, por serem algumas das maiores existentes e as principais às quais o autor desta dissertação teve acesso. A *string* de busca definida foi aplicada em cada uma das bases escolhidas, resultando em um total de 270 trabalhos encontrados (121 na Scopus e 149 na Engineering Village).

Para cada um dos trabalhos retornados por cada base de busca, foi realizada a leitura do título e *abstract*, eliminando-se aqueles que não tinham relação com a detecção de *usability smells*. Assim, 11 trabalhos foram selecionados para a leitura completa. Entre eles, 7 trabalhos foram identificados como diferentes fases da evolução de uma mesma pesquisa, portanto, apenas o mais completo desses será detalhado neste capítulo. A lista completa de trabalhos identificados nesta pesquisa sistemática está presente no Apêndice A.

O procedimento de pesquisa de trabalhos relacionados utilizado nesta dissertação foi inspirado pelos *guidelines* propostos por Kitchenham e Charters (2007) e Petersen et al. (2008). Contudo, o mesmo não segue o rigor e sistematicidade necessários a mapeamentos e revisões da literatura, além de ter sido realizado por um único pesquisador. Também não foram usadas questões de pesquisa para a extração de informações dos trabalhos selecionados, apenas a leitura informal. Em adição, os únicos critérios de inclusão/exclusão utilizados foram o trabalho estar relacionado à detecção de *usability smells* e ter sido publicado até o ano de 2017.

2.2 Abordagens de Auxílio à Testes de Usabilidade

A UsAGE (UEHLING; WOLF, 1995) foi uma das abordagens pioneiras a propor algum tipo de auxílio ao teste laboratorial. Ela é baseada na utilização de diferentes técnicas para comparar o fluxo de ações de usuários "novatos" e "experientes", gravados através da interação com interfaces desenvolvidas com apoio do sistema TAE Plus (SZCZUR; SHEPPARD, 1993). A ideia dessa comparação é a de que os desvios entre esses dois tipos de "caminhos" percorridos pelo usuário poderiam indicar locais da interface contendo problemas de usabilidade.

Várias outras ferramentas, como a USABILICS (VASCONCELOS; BALDOCHI JR., 2012), a WELFIT (SANTANA; BARANAUSKAS, 2015) e a UseSkill *Control* (SOUZA et al., 2015), similarmente utilizam a ideia de comparação entre diferentes tipos de fluxo para auxiliarem a execução do teste, cada uma com suas próprias particularidades. Outras formas de análise que utilizam como fonte os *logs* capturados de usuários também são bastante utilizadas (POUR; CALVO, 2011; BURZACCA; PATERNò, 2013; NEBELING; SPEICHER; NORRIE, 2013).

Além da comparação entre fluxos, outras técnicas como Inteligência Artificial (XU; XU, 2007) e Mineração de Dados (VARGAS; WEFFERS; ROCHA, 2010; VARGAS;

WEFFERS; ROCHA, 2011) também foram utilizadas com o intuito de facilitar a descoberta de problemas de usabilidade. Entretanto, cada uma dessas técnicas possui suas dificuldades de adoção próprias, como utilização complexa, resultados artificiais e pouco realistas, baixa praticidade, etc.

Apesar dos *refactorings* destinados à aplicações Web já terem sido utilizados anteriormente com outros propósitos, como aderência à padrões de *design* (GARRIDO; ROSSI; DISTANTE, 2007; OLSINA et al., 2007; OLSINA et al., 2008) e melhora da acessibilidade (MEDINA et al., 2010), a primeira ideia sobre a utilização de *refactorings* com foco em usabilidade foi proposta por Garrido, Rossi e Distante (2011). Esse trabalho também traz a primeira menção à *usability smells*, ainda que seja proposta apenas uma forma de detecção manual para os mesmos.

Almeida (2014), similarmente, apresenta uma abordagem para a detecção manual de *usability smells*. Contudo, desde então, todos os trabalhos relacionados com abordagens manuais acabaram evoluindo em ferramentas automáticas, além da proposição de novas abordagens, que serão discutidas na seção 2.3.

2.2.1 UseSkill

Com o intuito de reduzir os custos e a complexidade envolvidos em avaliações de usabilidade de sistemas Web foi desenvolvida a ferramenta UseSkill (SOUZA et al., 2016b). Inicialmente foi proposta visando auxiliar a realização de testes de usabilidade remotos apenas em contextos controlados, necessitando da definição de roteiros, tarefas e questionários. Esse módulo foi chamado de UseSkill *Control* (USC) (SOUZA et al., 2015).

Entretanto, convidar usuários, criar roteiros e preparar o sistema para ser testado envolvem custos e complexidade logística, logo, surgiu a necessidade de criar-se uma nova abordagem. Para implementar essa nova abordagem proposta foi desenvolvido o módulo UseSkill *OnTheFly* (USOTF) (SOUZA et al., 2016a), que contempla avaliações de usuários em seu ambiente de produção, executando suas atividades do dia-a-dia.

A abordagem implementada no módulo USOTF baseia-se na comparação entre utilizações “boas” e “ruis” do sistema. A ideia por trás dessa comparação é identificar quais partes das funcionalidades influenciam negativamente na utilização dos usuários e os fazem divergir entre “bons” e “ruins”, apontando possíveis pontos problemáticos. Essa abordagem foi dividida em quatro etapas: “Captura de Ações”, “Preparação de Dados”, “Análise de Dados” e “Geração de Relatórios”. A Figura 1 apresenta essas etapas da abordagem.

A Captura de Ações ocorre por meio de um componente desenvolvido em *JavaScript* que deve ser inserido no código-fonte da aplicação. Apesar de algumas informações serem obrigatórias (para diferenciar as ações), ele é personalizável, permitindo capturar diversos tipos de informações que podem ser úteis durante a análise.

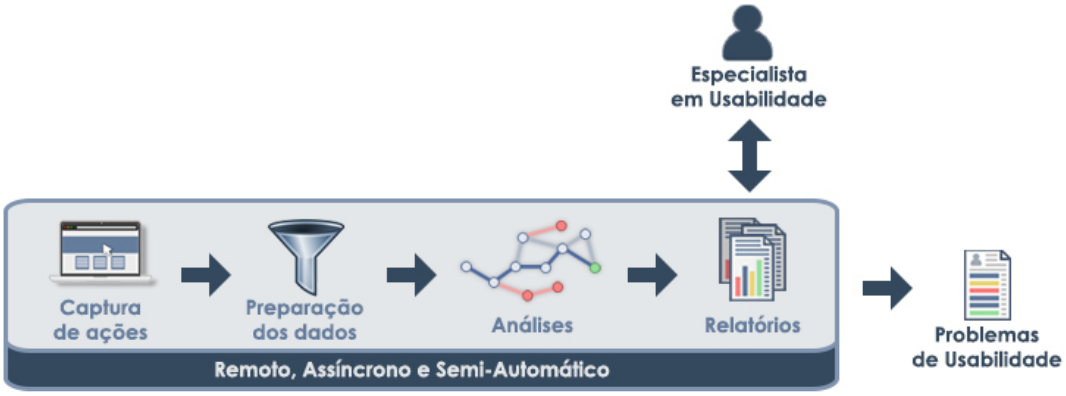


Figura 1 – Abordagem proposta pela *UseSkill OnTheFly*. Fonte: Souza et al. (2016a).

A Preparação de Dados baseia-se nas definições do pré-processamento de dados em *Web Usage Mining* (MOBASHER, 2007 apud SOUZA et al., 2016a). A primeira fase da preparação é a identificação de quais ações capturadas fazem parte da funcionalidade a ser analisada. Essa delimitação dos dados é importante para que cada funcionalidade possa ser analisada isoladamente. Em seguida ocorre a limpeza dos dados, removendo tipos de ações indesejadas.

Com os dados delimitados e limpos, ocorre a identificação das sessões de uso. Cada sessão representa um conjunto de ações realizadas por determinado usuário ao utilizar uma funcionalidade de um sistema. Seu conceito é semelhante à de uma utilização da funcionalidade. Um usuário, por exemplo, pode utilizar diversas vezes a mesma funcionalidade do sistema, gerando assim diversas sessões de uso.

Com as sessões capturadas, inicia-se a Análise de Dados, cujo primeiro passo é calcular métricas para medir a qualidade de uso de cada sessão. A ferramenta realiza o cálculo de duas métricas: eficácia e eficiência. Em seguida, de acordo com as métricas calculadas para cada uma das sessões, elas são classificadas como “boas” ou “ruins”. Caso a sessão tenha bons índices de eficácia e eficiência, ela será classificada como uma “boa” sessão. Caso contrário, se os índices forem baixos, a sessão será classificada como “ruim”.

Com as sessões classificadas, a ideia é agrupá-las em: Grupo de Sessões Referência (GSR), contendo as utilizações das funcionalidades do sistema de maneira esperada; ou Grupo das Demais Sessões (GDS), que não lograram êxito, realizaram ações demasiadamente ou demoraram muito para finalizar a sessão. Após o agrupamento das sessões em GSR e GDS é possível comparar tais grupos a fim de encontrar diferenças entre eles. Essas comparações servem como base para a classificação de cada uma das ações contidas nas sessões.

As ações que foram mais frequentes no GSR são classificadas como “Ações Obrigatórias”(AO), ou seja, passos que os usuários devem realizar para utilizar a funcionalidade corretamente. As demais ações contidas no GSR e que não foram classificadas como AO,

são as "Ações Corretas" (AC). As ações mais frequentes no GDS e que não estão entre as mais frequentes no GSR são as "Ações Problemáticas" (AP). Por fim, as ações contidas no GDS, que não fazem parte do GSR e que não foram classificadas como AP, são consideradas "Ações Alertas" (AA).

A classificação das ações é uma etapa importante para gerar relatórios que apontem para as partes de funcionalidades com maior possibilidade de possuírem problemas de usabilidade. As ações AP e AA servem para dar indícios de onde estão os locais problemáticos e as métricas apontam quais sessões enfrentaram mais dificuldades.

A partir das métricas e classificações realizadas, ocorre a Geração de Relatórios que apoiam a análise e interpretação dos dados por parte de especialistas em usabilidade. A proposta baseia-se na possibilidade de ter uma visão geral da usabilidade e ao mesmo tempo permitir análises aprofundadas em determinadas utilizações.

A abordagem utiliza listas contendo as ações realizadas sequencialmente e possibilita selecionar ações para visualizar as informações capturadas, como tipo de ação, elemento, local, horário e usuário. Com isso é possível, por exemplo, verificar quanto tempo o usuário demorou entre uma ação e outra, ou identificar quais elementos receberam mais ações repetidamente.

A ferramenta também gera grafos que permitem aos especialistas, através de uma análise rápida, terem noções gerais da usabilidade do sistema. Os nós dos grafos representam as ações realizadas, sendo que os nós redondos correspondem às ações mais frequentes e os quadrados são as ações que não estão entre as mais frequentes. As arestas do grafo apontam quais caminhos foram percorridos pelos usuários, as cores dos nós indicam a classificação de cada ação, a largura das arestas apontam quais caminhos foram mais percorridos e o tamanho de cada nó (diâmetro ou medida do lado) representa quais ações foram mais realizadas.

A Figura 2 exemplifica um grafo gerado pela UseSkill, que apresenta uma sessão de uso. Nele há nós e arestas maiores em relação aos demais, pois foram mais realizados. As cores representam a classificação das ações, sendo as azuis AO, as verdes AC e as amarelas AA. É perceptível também que há ciclos no grafo, apresentando conjuntos de ações que foram repetidas. A existência de ciclos, o tamanho dos nós e a classificação das ações são fundamentais para apoiar uma visualização geral da qualidade de uso da funcionalidade.

A ferramenta USOTF não detecta automaticamente problemas de usabilidade ou *usability smells*, necessitando do auxílio de um especialista para a sua utilização. Entretanto ela realiza a captura de eventos de baixo nível em contexto de produção, conforme o proposto pela abordagem deste trabalho, e a geração de métricas valiosas que podem ser usadas para a detecção de problemas. Por esse motivo, a abordagem proposta nesta dissertação utiliza a ferramenta UseSkill como parte de seu processo de captura e análise

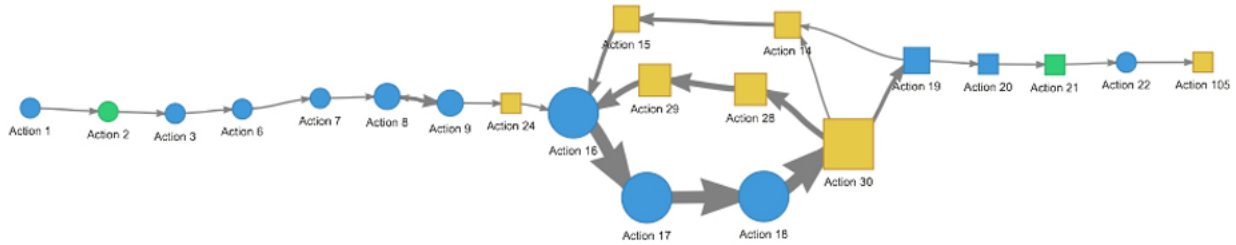


Figura 2 – Grafo exemplificando uma sessão de uso. Fonte: Souza et al. (2016a).

de dados, ao mesmo tempo em que a estende, adicionando a capacidade de detecção automática de *usability smells*.

2.3 Abordagens de Detecção de *Usability Smells*

2.3.1 *Workflow* para a Mineração de Padrões de Uso Frequentes

A abordagem apresentada por Damevski et al. (2017) propõe um *workflow* exploratório para a mineração de padrões de uso frequentes de interações a partir de um grande repositório de dados de uso da IDE (*Integrated Development Environment*) Visual Studio. Utilizando diferentes técnicas de mineração de dados, é possível extrair os comportamentos mais comuns exibidos pelos desenvolvedores e, através da análise desses, identificar *usability smells* na IDE.

Essa abordagem, contudo, não apresenta nenhuma estratégia para a detecção automática de *usability smells*, nem foi consolidada na forma de uma ferramenta. Os *smells* são determinados a partir da análise manual dos padrões minerados, por especialistas, e foram definidos apenas posteriormente à ela. Os *smells* encontrados não receberam nenhuma denominação específica, sendo eles:

- **Retorno dos resultados de pesquisa subótimo**: os desenvolvedores se movem bem rápido entre cada clique e a visualização do resultado é relativamente demorada;
- **Comportamento repetitivo dos desenvolvedores**: apenas um conjunto específico de comandos é usado, ainda que comandos mais eficientes estejam disponíveis;
- **Suporte do conjunto de trabalho não ótimo**: muitas vezes os arquivos são abertos sem edição subsequente;

A abordagem proposta por Damevski et al. (2017) é a abordagem de detecção de *smells* que mais se distancia da proposta desta dissertação. De fato, esse trabalho usa mais frequentemente o termo *usage smells*, em vez de *usability smells*, para se referir aos indicativos encontrados. Contudo, a ideia do que eles representam se mantém a mesma.

Essa abordagem é voltada especificamente para IDEs, mas tem como vantagem o fato de exigir um conjunto da dados de entrada bem simples de obter. A abordagem proposta exige que os dados de entrada sejam capturados através de um *script* que deve ser inserido no código fonte da aplicação avaliada. Contudo, ela é voltada para o contexto Web, que é mais amplo e afeta bem mais usuários, define estratégias para a detecção automática de *smells* e foi consolidada na forma de extensão de uma ferramenta.

Apesar das diferenças de contexto e objetivos, a abordagem de Damevski et al. (2017) apresenta uma boa alternativa para a proposição de *usability smells*, ou seja, a indicação de problemas, por parte dos especialistas, sobre determinados comportamentos dos usuários é uma boa fonte para definir novos *smells*. De fato, essa ideia foi parcialmente utilizada na definição de alguns dos *smells* propostos nesta dissertação, conforme será apresentado no Capítulo 3.

2.3.2 AutoQUEST

A abordagem de Harms e Grabowski (2014) utiliza árvores de tarefas, geradas a partir da captura de dados de utilização do usuário, para identificar *usability smells*. Uma tarefa, nesse caso, pode ser entendida como uma sequência de ações do usuário que resulta no cumprimento de uma determinada meta (o *login* em uma aplicação, por exemplo). Cada tarefa, por sua vez, pode conter uma série de sub-tarefas internas como parte de sua execução (uma tarefa de transferência bancária online, por exemplo, pode ter como sub-tarefa a realização do *login*).

Cada tarefa também possui um "relacionamento temporal", que pode ser do tipo "sequência" ou "iteração", sendo as tarefas com esse tipo de relacionamento referidas apenas pelo respectivo tipo. No caso da sequência, uma tarefa tem um ou mais filhos e eles são executados na sua respectiva ordem. No caso da iteração, uma tarefa tem exatamente um filho, que é executado zero ou mais vezes.

Para cada um dos *smells* catalogados, os autores proveem uma fundação (fonte(s) de onde ele foi retirado ou inspirado), descrição do comportamento esperado dos usuários (comportamento que indica o aparecimento do *smell*) e descrição das expectativas dos autores em relação às árvores de tarefas (sequências de ações que indicam o aparecimento do *smell*). Essa abordagem é capaz de detectar 4 diferentes *usability smells*, sendo eles:

- ***Missing Feedback***: ao utilizar uma aplicação, os usuários precisam de *feedback* para as ações realizadas (POLSON et al., 1992; FERRÉ et al., 2001; NORMAN,

2002; BALBO et al., 2005; HARMS; GRABOWSKI, 2014), caso contrário, não sabem se sua ação teve algum efeito e foi processada pelo sistema. Os usuários tendem a repetir uma ação se eles não veem nenhum *feedback*, logo, ações repetidas levam a iterações nas árvores de tarefas. Apesar disso, nem todas as iterações de ações são indicativas de *smells*, portanto, apenas um subgrupo de iterações de ações é considerado;

- ***Important Task***: tarefas realizadas regularmente devem ser executadas com o número mínimo de ações possível (LECEROF; PATERNò, 1998; TIDWELL, 2010; HARMS; GRABOWSKI, 2014), portanto, se aplicável, devem ser auxiliadas por automação. Se os usuários realizam uma tarefa regularmente, eles executam sempre as mesmas ações similares, comportamento que é refletido nos nós das árvores de tarefas geradas;

- ***Required Inefficient Actions***: algumas ações têm pouco ou nenhum efeito na realização de uma tarefa, mas também não impedem a sua execução (PATERNÒ; PIRUZZA; SANTORO, 2005 apud HARMS; GRABOWSKI, 2014). O número dessas "ações ineficientes" deve ser minimizado para aumentar a eficiência dos usuários. Os usuários executam ações ineficientes durante o uso normal da aplicação, e, como as árvores de tarefas são geradas baseadas nas ações capturadas, elas irão incluir as ações ineficientes;

- ***High Website Element Distance***: os elementos da interface de uma aplicação Web necessários para a execução de uma tarefa devem ser colocados tão próximo quanto possível para assegurar a execução eficiente da tarefa (LECEROF; PATERNò, 1998 apud HARMS; GRABOWSKI, 2014). Os usuários utilizam os elementos da interface de acordo com o que se adapta melhor às tarefas para as quais eles usam a aplicação, e as árvores de tarefas mostram a sequência em que os elementos da interface são usados.

A abordagem foi implementada baseada na suíte de ferramentas AutoQUEST (HERBOLD; HARMS, 2013 apud HARMS; GRABOWSKI, 2014). Entre outras funções, a AutoQUEST permite a captura de ações de usuários em diferentes plataformas. Para isso, cada página da aplicação deve incluir um *JavaScript* provido por ela. As ações são, assim, registradas e enviadas a um servidor dedicado que as armazena para análise posterior. A AutoQUEST também provê mecanismos para a geração de árvores de tarefas baseada nas ações capturadas dos usuários. Ela foi utilizada e expandida pela abordagem, com a implementação da detecção dos *usability smells* apresentados, por meio de diferentes algoritmos.

A abordagem proposta por Harms e Grabowski (2014) é similar a apresentada neste trabalho, na forma de captura de dados e no intuito de detectar *usability smells*. Apesar de

ter sido capaz de ajudar a identificar problemas de usabilidade concretos, essa abordagem não considera, como parâmetro de detecção, a quantidade de vezes que cada *smell* aparece, o que gera uma grande quantidade de falsos positivos e detecções redundantes. Além disso, parte da análise dos dados e identificação dos *smells* nas árvores de tarefas ainda tem de ser feita de forma manual.

A abordagem proposta neste trabalho considera a quantidade de vezes em que cada *smell* aparece como parte da detecção para evitar redundâncias e torná-la mais precisa. Ela também exige que um avaliador constate se os *smells* detectados indicam problemas reais de usabilidade, contudo, uma vez configurada a captura, a detecção de *smells* é feita de forma completamente automática. Além disso, são fornecidos ao avaliador vários relatórios simplificados, na forma de grafos e tabelas, para auxiliá-lo nessa tarefa.

2.3.3 GUISurfer

Almeida et al. (2015) propuseram um catálogo composto por seis *usability smells* adaptados a partir dos *code smells* de Fowler (FOWLER; BECK, 1999). Evidentemente, nem todos os *smells* de código possuem uma correspondência no domínio da usabilidade. Contudo, os autores também utilizaram como base o trabalho de Hermans, Pinzger e Deursen (2012), que já havia conseguido adaptar, com sucesso, esses *smells* para o contexto das planilhas de cálculo. Consequentemente, os mesmos *smells* adaptados nesse trabalho foram utilizados, porém, considerando-os como definindo interdependências entre janelas em aplicações de software interativas.

Os *usability smells* catalogados foram divididos em 3 grupos: "implementação", "*design*" e "domínio". Presentes no grupo de implementação estão os relacionados à estrutura do código fonte da aplicação. Esse grupo é composto por dois *smells*:

- ***Shotgun Surgery***: esse *smell* trata do fato de as interfaces de usuário serem objeto de constante mudança, podendo a estrutura do código fonte impactar negativamente com a habilidade de mudar partes da IU. É normal para os programadores, por exemplo, utilizarem o mesmo formulário base em diferentes janelas. Conforme a quantidade de reuso aumenta, a dificuldade do programador de lembrar-se todos os lugares em que isso aconteceu também aumenta. Uma solução para isso é refatorar a definição do formulário em uma única classe ou método. Se diferentes janelas onde o formulário é usado pertencem ao mesmo domínio, a possibilidade de *redesign* da IU, de forma que um único par de janela/formulário seja usado (i.e. agrupar todas as janelas em uma), também deve ser considerada;
- ***Too Many Layers***: ocorre quando uma aplicação tem muitas janelas e, para realizar uma tarefa, os usuários precisam passar por quatro, cinco ou até mais delas. A necessidade de passar por todas essas janelas afeta severamente a performance

do usuário em completar uma tarefa. Uma possibilidade para eliminar o *Too Many Layers* é a criação de atalhos para todas as janelas relevantes, usando, por exemplo, um menu. Outra possibilidade é reunir todas as janelas com estrutura e informações comuns. Esses possíveis *refactorings* ajudam os usuários a navegar pela interface mais facilmente, sem excesso de informação e usando uma estruturação melhor.

Os *smells* no grupo de *design* capturam funcionalidades do *design* da IU. Esse grupo é composto pelos seguintes *smells*:

- ***Middle Man***: ocorre quando há ao menos duas janelas, e a primeira desnecessariamente delega informações ou competências a segunda. A anomalia acontece quando essa informação pode ser mostrada (ou a competência executada) pela primeira janela com vantagem, mas por alguma razão foi decidido delegá-la a outra. Um exemplo dessa ocorrência seria quando o usuário tenta submeter um formulário com algum campo vazio e uma nova janela aparece mostrando uma mensagem de erro. Uma ideia para essa refatoração é, caso a janela que delega essa tarefa possa executá-la, manter essa tarefa nela, evitando que usuário seja distraído por outra janela, já que o foco principal de atenção permanece na janela principal;
- ***Information Overload***: ocorre quando muita informação é apresentada na tela ao mesmo tempo. Esse é o *usability smell* mais típico. Ele pode ser visto, por exemplo, em muitas aplicações Web e *mobile* que implementam sistemas de propaganda. Outro exemplo são aplicações que possuem componentes inúteis, como botões e *links* que não realizam nenhuma ação.

Finalmente o grupo de domínio consiste de *smells* associados a potenciais problemas na IU que estão relacionados ao domínio da aplicação. Esse grupo aparece em janelas com o mesmo domínio, isto é, que contêm informações relacionadas. São eles:

- ***Inappropriate Intimacy***: ocorre quando duas ou mais janelas que pertencem ao mesmo domínio e têm funcionalidades que dependem uma da outra são acessadas por caminhos diferentes. A ideia para essa refatoração é, em primeiro lugar, validar que todas as janelas pertencem ao mesmo domínio e, se essa validação ocorrer, agrupar todas as janelas em uma para facilitar aos usuários a realização das tarefas que elas contêm, acessando apenas um caminho;
- ***Feature Envy***: acontece quando uma janela tem ao menos uma tarefa que pertence a outra. Por exemplo, considere duas janelas, a primeira usada para gerenciar contas bancárias e a segunda para mostrar informações de saldo de contas. A segunda janela tem a possibilidade de introduzir novos clientes. Essa tarefa não pertence a essa

janela, pois o seu propósito é mostrar informações, e não gerenciar as contas dos clientes. A solução para esse *smell* é entender o domínio e transferir a tarefa errada para o seu lugar apropriado.

O *framework* GUISurfer (SILVA et al., 2010 apud ALMEIDA et al., 2015) provê bibliotecas e ferramentas que suportam a análise automática de sistemas interativos. A ferramenta permite a extração de modelos comportamentais de GUI (*Graphical User Interface*, ou Interface Gráfica do Usuário) realistas a partir do código fonte de uma aplicação interativa. Almeida et al. (2015) estenderam essa ferramenta para permitir a detecção dos *usability smells* do seu catálogo, no modelo comportamental gerado.

Para identificar os *usability smells* no modelo comportamental de GUI gerado, são aplicados diferentes algoritmos e métricas baseados em grafos, que são capazes de computar tais *smells*. Como resultado, a GUISurfer gera modelos com os *usability smells* marcados na representação visual. Os modelos gerados também foram estendidos para que, quando o usuário selecione uma região marcada, uma notificação identificando o *smell* seja automaticamente exibida.

A ferramenta apresentada difere da proposta neste trabalho por considerar apenas o código fonte da aplicação como fonte de informação para a detecção automática dos *smells*. O conjunto de *usability smells* que pode ser detectado sem a análise da interação dos usuários propriamente dita é bem limitado. Além disso, o contexto abordado pela ferramenta é o de aplicações *Desktop*. Apesar desse contexto englobar um grande número de aplicações, a Web é muito mais utilizada para promover a interação entre os usuários e as grandes empresas, por ser de acesso mais fácil, simples e portátil. Assim, a abordagem proposta utiliza o contexto Web, por considerá-lo mais crítico e difundido.

2.3.4 MUSE

A MUSE (*Mobile Usability Smell Evaluator*) (PATERNò; SCHIAVONE; CONTE, 2017) é uma ferramenta de avaliação de usabilidade baseada em *proxy* capaz de gravar o comportamento do usuário ao interagir com qualquer aplicação Web através de dispositivos móveis ou *Desktop*. Os dados de interação do usuário são capturados através de um *script* de captura injetado na página Web por meio de um servidor de *proxy*.

A definição de *usability smells* na ferramenta é feita através de uma linguagem baseada em XML, indicando-se a sequência de ações que representa a ocorrência de cada *smell*. Assim, a detecção de *smells* é feita checando-se, entre as sequências de ações capturadas dos usuários, pela presença de padrões de eventos que representam um ou mais *usability smells*. Paternò, Schiavone e Conte (2017) utilizaram como fonte trabalhos relacionados, informações providas por softwares comerciais, estudos significativos à respeito

de usabilidade móvel e a experiência pessoal para estabelecer um conjunto de 6 *smells*, sendo eles:

- ***Too Small or Close Elements***: esse *smell* é caracterizado pela presença de elementos da página Web que são excessivamente próximos uns dos outros. Campos de formulário posicionados muito proximamente, ou muito pequenos e/ou botões de seleção insuficientemente espaçados são alguns exemplos. Para visualizar o conteúdo efetivamente, o usuário é forçado a executar uma série de ações complexas para redimensionar o conteúdo;

- ***Too Close Links***: representa uma variante do caso anterior, mas é identificado por uma performance diferente do usuário. Ele envolve a presença de elementos muito próximos cuja interação ativa o carregamento de uma nova página Web. Um exemplo é uma série de links com espaçamento reduzido ou um botão de submeter formulário insuficientemente distanciado dos outros elementos. Nesse caso, os usuários podem carregar outra página por engano, e são forçados a refazer seus passos anteriores;

- ***Distant Content***: diz respeito à presença de conteúdos relacionados da página Web posicionados muito longe uns dos outros, e cuja exibição ou interação é crucial para a execução correta de algumas tarefas. O usuário é forçado a executar uma grande quantidade de rolagens de página para baixo e para cima;

- ***Too Small Section***: uma seção cujo tamanho aparece muito pequeno requer ações de ampliação específicas. O comportamento correspondente esperado é ampliar a seção através de um toque duplo (um gesto que funciona como atalho aumentar o zoom em muitos dispositivos móveis), e então continuar a atividade. Essas ações permitem aos usuários facilitarem o desempenho da tarefa;

- ***Bad Readability***: esse *smell* diz respeito às dificuldades encontradas pelo usuário quando está lendo conteúdo textual. Esse caso é detectado quando há interação com blocos de texto cujo tamanho da fonte ou o espaçamento é muito pequeno. O usuário executa uma série de ações com o objetivo de otimizar o tamanho da fonte e/ou a localização do bloco de texto para facilitar a leitura, tais como sequências de ações *pinch*[1] e *pan*[2], mas não seguidas de interações tais como toques pois a tarefa é de leitura;

- ***Long Forms***: esse *smell* é caracterizado pela presença de um alto número de interações com campos de texto, considerada excessiva para a boa usabilidade em dispositivos móveis.

[1] Gesto de movimentar os dedos de forma similar a uma pinça na tela, geralmente utilizado para aumentar ou diminuir o nível de zoom.

[2] Gesto de arrastar o dedo pela tela, geralmente usado para navegar através de conteúdos maiores que o tamanho da mesma.

A ferramenta possibilita a criação de sessões de avaliação de usabilidade com a indicação das tarefas que as compõem. Os dados capturados durante as sessões de teste com usuários podem ser visualizadas através de um módulo de análise, que provê uma visão geral dos *logs* coletados, associada com a visualização de uma linha do tempo interativa, que indica onde os *smells* ocorreram.

A abordagem proposta difere da ferramenta em questão principalmente devido ao contexto e aos requerimentos necessários para a sua utilização. A MUSE utiliza um servidor *proxy* para injetar o *script* de captura na aplicação testada, o que é pouco intrusivo, mas implica em um certo esforço de configuração. A abordagem proposta precisa que o *script* seja inserido manualmente no código fonte da aplicação, pelo desenvolvedor da mesma, o que requer um nível maior de acesso, mas não exige o uso de nenhum equipamento extra.

Também é possível perceber, pela definição dos *smells*, que a MUSE é focada na avaliação da usabilidade de aplicações Web para dispositivos móveis. A abordagem proposta é focada apenas no ambiente Web *Desktop*, não permitindo a captura de eventos gerados exclusivamente por dispositivos móveis, como toques e *pinchs*. Contudo, ela pode ser expandida, em versões futuras, para incorporar esse tipo de ambiente.

Por fim, a definição de *smells* na MUSE é feita através da especificação de uma sequência de ações em linguagem XML. Essa forma de definição é bem mais simples que a utilizada pela abordagem proposta, que exige a definição programática dos mesmos. Contudo, isso limita a detecção de *smells* à identificação de sequências de ações específicas, não permitindo que outros aspectos sejam analisados.

2.3.5 Usability Smells Finder

A estratégia automatizada de reconhecimento de *smells* proposta por Grigera et al. (2017) é baseada na análise de eventos de UI (*User Interaction* ou Interação do Usuário). Contudo, eles estendem os trabalhos anteriores na área, vinculando eventos de UI específicos a *usability smells*, definindo novos *usability smells* e reportando *usability smells on-the-fly*. Os autores implementaram essa abordagem em uma ferramenta chamada USF (*Usability Smells Finder*). A ferramenta pode ser usada como um serviço, com esforço mínimo de configuração, e é capaz de prover conselhos atualizados para aplicações Web implantadas. Além disso, ela é implementada de forma a permitir a extensão das estratégias de detecção de *usability smells*.

Baseados na literatura anterior sobre análise de *logs* de eventos de UI, os autores também definiram um catálogo de *usability smells* a partir da abstração de padrões de eventos de usuário que foram mostrados como causadores de interação problemática. Com isso, eles são capazes de definir nome, descrição, exemplo e *refactoring* para cada um dos *smells*. Esses *smells* são:

- ***Undescriptive Element***: esse *smell* aparece quando muitos usuários tentam conseguir uma *tooltip* de um elemento da interface. Isso pode indicar que o elemento não é auto-descritivo o suficiente;

- ***Misleading Link***: esse *smell* é similar ao *Undescriptive Element*, mas específico aos *links*. Além das tentativas de *tooltip* em *links*, esse *smell* captura sequências de eventos onde muitos usuários navegam em um *link* apenas para retornarem logo após, indicando que, mais provavelmente, os conteúdos vinculados não representam corretamente o nome do *link*;

- ***No Processing Page***: esse *smell* é detectado quando os usuários têm de esperar muito tempo pelo carregamento de uma página sem *feedback*. O limiar de espera pretende capturar o momento em que os usuários perdem o interesse;

- ***Free Input for Limited Values***: esse *smell* é disparado quando uma entrada de texto padrão é usada para a entrada de dados de um conjunto limitado de valores. Os usuários são forçados a digitar o texto completo quando, de fato, as opções são restritas. Esse *smell* também pode aparecer quando as opções não são restritas, mas, na verdade, há apenas um pequeno conjunto de escolhas bem populares. Nesse caso, ao menos uma função de preenchimento automático deveria ajudar os usuários a escreverem menos texto, sentindo-se mais seguros sobre a opção entrada;

- ***Unformatted Input***: esse *smell* indica que uma entrada de texto padrão está sendo usada para entrar dados em um formato determinado, como datas, números de telefone ou CEPs. Nesses casos, o formulário poderia ajudar os usuários a formatar os dados. Esse smell é detectado checando-se a correspondência dos dados entrados com expressões regulares (uma para cada formato);

- ***Short Input***: esse *smell* é detectado quando um campo de entrada de texto não é grande o suficiente para corresponder ao tamanho dos textos mais frequentemente usados como entrada. Campos de entrada de texto que são muito maiores do que o necessário também podem confundir os usuários;

- ***Unnecessary Bulk Action***: é comum para aplicações Web que mostram listas de itens (como produtos ou mensagens) oferecerem a possibilidade de ações em grupo. Os usuários tipicamente realizam essas ações selecionando um grupo de itens usando *checkboxes* e depois aplicando uma ação (e.g. “Apagar”, “Mover”, etc.). Apesar de essa interação funcionar bem quando aplicada a grupos de itens, ela é demorada para aplicar a um único item. Quando esse é o caso mais frequente e os usuários não têm uma forma mais rápida de aplicar essas ações, o *smell Unnecessary Bulk Action* é detectado;

- ***Overlooked Content***: alguns conteúdos geralmente são passados muito rapidamente pelos usuários, então, é provável que eles não os leiam/vejam. Além disso, eles geralmente param na mesma área (posição y), logo, possivelmente o conteúdo alcançado poderia ser mais fácil de alcançar se estivesse mais próximo ao topo. Além de indicar um potencial problema de usabilidade, esse *smell* provê informações valiosas, já que o conteúdo negligenciado é considerado importante pelo dono da aplicação Web;

- ***Distant Content***: esse *usability smell* detecta caminhos de navegação repetidos onde os usuários ficam por apenas um curto tempo nos nós intermediários. O raciocínio por trás do *smell* é que os usuários estão procurando pelos conteúdos no nó final, mas um longo caminho é requerido para alcançá-lo. O *smell Distant Content* indica que o caminho mais seguido de um nó A a um nó B pode ser muito longo;

- ***No Client Validation***: esse *smell* detecta formulários que tem uma alta taxa de rejeição, onde a validação ocorre do lado do servidor. Esse cenário é detectado mantendo-se o rastro de uma nova requisição, e verificando pela presença do mesmo formulário após a requisição;

- ***Late Validation***: esse *smell* é essencialmente igual ao anterior, *No Client Validation*, com a diferença de ser específico para o cenário onde a validação ocorre após o botão "Submeter" ser clicado, mas a submissão não acontece. Isso geralmente quer dizer que há validação do lado do cliente, mas a mesma poderia ser melhorada por validação em linha;

- ***Abandoned Form***: esse *smell* é detectado quando a taxa de abandono de um formulário excede um dado limiar, alertando o avaliador sobre um formulário que provavelmente é muito complexo e desencoraja os usuários a preenchê-lo;

- ***Scarce Search Results***: esse *smell* detecta quando um formulário de busca não traz resultados na maior parte das vezes. Ele também armazena as buscas sem sucesso mais populares para apresentar ao avaliador um ranking;

- ***Useless Search Results***: como uma variante do listado previamente, *Scarce Search Results*, esse *smell* detecta o que acontece após um formulário de busca trazer resultados. Se os usuários raramente clicam em um desses resultados, então pode ser que tais resultados, mesmo aparecendo, não são o que os usuários estão buscando. Algumas vezes, entretanto, a página de resultados pode ser apenas informativa o suficiente, de modo que mais cliques são desnecessários. Nesses casos, esse *smell* pode ser ignorado;

- ***Wrong Default Value***: esse *smell* detecta quando a escolha mais popular de um *radio button* ou *select box* não corresponde ao seu valor padrão. Preencher um formulário pode ser tedioso, então, ter o mínimo de questões possível para preencher aumenta a probabilidade de os usuários o completarem;
- ***Unresponsive Element***: esse *smell* é detectado quando um elemento é geralmente clicado pelos usuários, mas não dispara nenhuma ação. Isso acontece quando tais elementos têm uma aparência sugestiva. Elementos típicos onde esse smell aparece incluem listas de fotos de produtos, cabeçalhos de aplicações Web, *labels* de *checkbox*/*radio button* e textos sublinhados/destacados.

De todos os trabalhos relacionados, essa é a abordagem que mais se assemelha a proposta neste trabalho. As diferenças principais estão no objetivo de cada trabalho e como esse objetivo é refletido na implementação. No caso da USF, os autores priorizam o nível de esforço (tentando reduzi-lo) e o desempenho (em termos de tempo de processamento). Logo, os eventos capturados são previamente filtrados e agrupados em eventos de alto nível, e o processamento ocorre de maneira contínua. Além disso, a única configuração exigida é a inserção de um *script* de captura no código fonte da aplicação.

Neste trabalho, prioriza-se a abrangência e precisão na detecção dos *smells*, assim, uma etapa adicional de configuração é necessária (definição de tarefas). Apesar disso, a configuração adicional é bem simples de ser realizada (exigindo apenas que se conheça a aplicação), e expande o universo de problemas detectáveis àqueles associados a tarefas específicas. Em adição, o processamento de eventos é realizado com aqueles de baixo nível e apenas quando solicitado explicitamente pelo avaliador, o que é mais demorado, mas nos permite obter resultados mais precisos sobre as ações dos usuários, além de ocupar o servidor por menos tempo.

2.4 Considerações Finais

Este capítulo apresentou um resumo sobre os principais trabalhos relacionados a abordagem proposta nesta dissertação. Os trabalhos de Harms e Grabowski (2014), Almeida et al. (2015) e Grigera et al. (2017) foram de extrema importância para fundamentar não só a teoria a respeito de *usability smells*, mas também a abordagem que será utilizada neste trabalho. O trabalho de Souza (2016), apesar de não abordar a detecção de *smells*, também trouxe importantes colaborações teóricas, além de ter sido integrado como parte da própria abordagem deste trabalho.

3 Abordagem Proposta

Este capítulo apresenta o detalhamento sobre a implementação e execução da abordagem proposta nesta dissertação, que busca detectar *usability smells* e sugerir *refactorings*[1] para os mesmos. Cada *smell* utilizado possui uma definição baseada em algum princípio de usabilidade, bem como uma estratégia de detecção[2] automática associada ao mesmo. Essas estratégias de detecção foram implementadas na forma de uma aplicação Web, estendendo as funcionalidades da ferramenta UseSkill (SOUZA et al., 2016a).

A utilização da funcionalidade de detecção de *smells* requer a execução de uma série de passos bem definidos, que podem ser agrupados em 3 etapas principais: **Captura de Dados**, **Análise de Dados** e **Geração de Relatórios**. Um resumo sobre as atividades realizadas durante cada uma dessas etapas é apresentado na Figura 3.

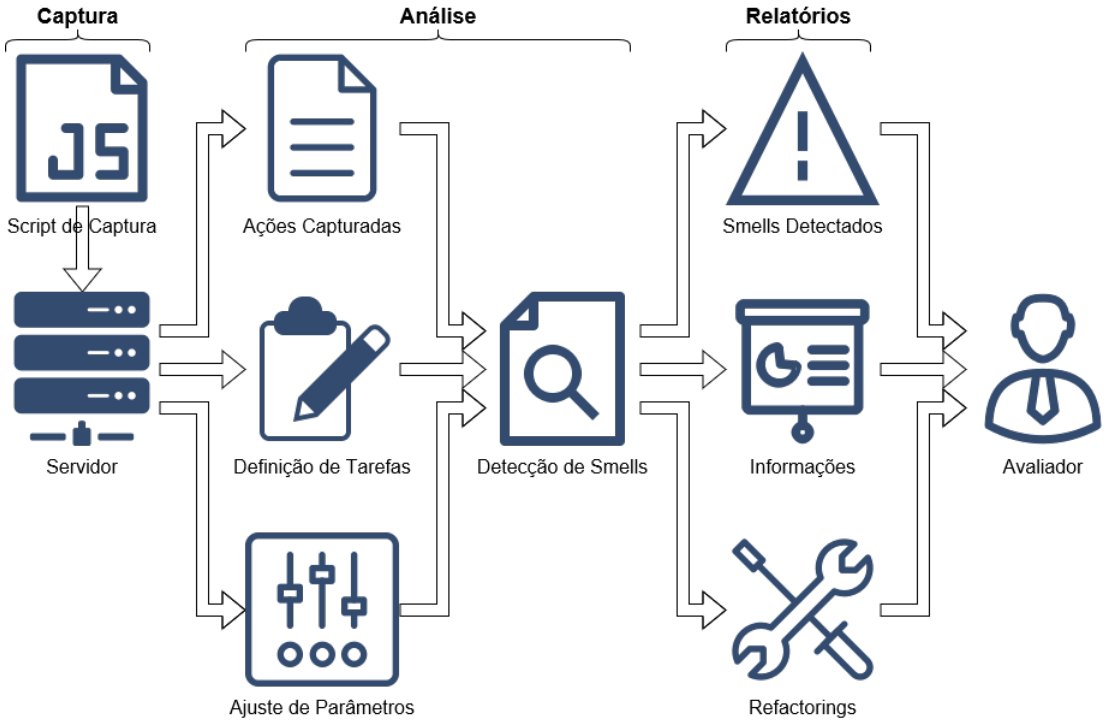


Figura 3 – Resumo da abordagem proposta. Fonte: Autor (2018).

[1] É importante destacar que os *refactorings* sugeridos neste trabalho não são tão diretos como os propostos por Fowler, uma vez que alguns dos *usability smells* propostos podem estar associados à mais de um problema de usabilidade. Assim, eles devem ser vistos mais como dicas do que ações unitárias.

[2] O termo "estratégia de detecção" foi introduzido neste trabalho para salientar o fato de que a detecção realizada não é uma tradução perfeita do conceito do *smell* correspondente, uma vez que esses não apresentam, em sua definição, valores exatos sobre o que representa um problema.

3.1 Etapas da Abordagem

3.1.1 Captura de Dados

Na primeira etapa, as ações dos usuários reais interagindo com o sistema precisam ser capturadas. Para que isso seja possível, a abordagem proposta prevê que um *script* de captura de ações deve ser inserido no código fonte da aplicação. Desse modo, é possível capturar ações mais detalhadas que não seriam possíveis com a captura de dados em servidor. Isso permite ter uma melhor noção do caminho percorrido pelo usuário e, consequentemente, ajuda a tornar a análise mais precisa.

Essa etapa precisa ser realizada manualmente, contudo, ela se limita à um conjunto de ações bem simples. O *script* é provido pela própria implementação da abordagem e precisa apenas ser ajustado para se adaptar às particularidades da aplicação testada. As ações capturadas são armazenadas na forma de *logs* e usadas como entrada para a próxima etapa da abordagem.

3.1.2 Análise de Dados

Na segunda etapa, antes que as ações sejam analisadas, é necessário realizar a **definição de tarefas**. Essa definição pode ser realizada por qualquer conhecedor da aplicação e consiste em apontar, para cada funcionalidade (tarefa) que deseja-se analisar, o conjunto de "ações iniciais" (ações pelas quais a tarefa pode ser iniciada) e "ações finais" (ações que indicam que o usuário chegou ao fim do fluxo de execução e conseguiu realizar a tarefa). Cada ação é unicamente identificada por uma associação entre o tipo da ação, URL onde ocorre e XPath (W3C, 1999) do elemento interagido. Assim, é necessário indicar cada um desses elementos para cada ação apontada nessa etapa.

A definição de tarefas permite que os dados capturados sejam agrupados na forma de sessões durante a busca por *smells*. Uma **sessão** corresponde a execução (ou tentativa de execução) de uma tarefa por um determinado usuário. A partir das sessões é possível saber qual tarefa o usuário estava executando, se ele conseguiu finalizá-la, em que ação específica ele teve dificuldades, entre outras informações valiosas que dão pistas sobre os pontos problemáticos da aplicação. Nessa etapa as sessões também são filtradas e classificadas, podendo ser de 4 tipos: "completa", "reinício", "limiar" ou "erro".

As sessões completas correspondem a utilizações corretas da funcionalidade conforme o esperado, ou seja, onde o usuário passou por ao menos um dos pontos iniciais e finais. Essas sessões ajudam, por exemplo, a identificar estatísticas sobre a utilização esperada (como tempo e quantidade de ações média), que podem ser comparadas com as demais sessões para a detecção de *smells*. A Figura 4 mostra um exemplo de sessão completa. Nesse grafo, o usuário começou pela ação inicial *Action 1* e terminou com ação

final *Action 79*

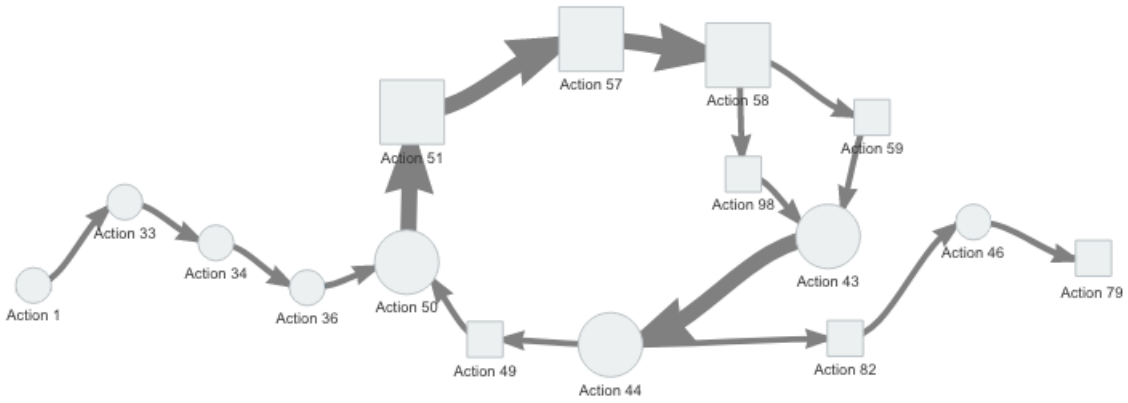


Figura 4 – Exemplo de grafo representando uma sessão completa. Fonte: Autor (2018).

As sessões de reinício representam utilizações onde o usuário começou a executar a tarefa por uma das ações iniciais, mas antes de finalizar reiniciou sua execução, ou seja, realizou alguma das ações iniciais novamente. Essas sessões indicam que houve algum problema durante a execução tarefa que obrigou-o a retornar ao início, podendo ser analisadas para identificar a causa desse problema que, possivelmente, está relacionado à usabilidade da aplicação. A Figura 5 mostra um exemplo de sessão de reinício, onde o usuário começou com a ação inicial *Action 1* e voltou a executar a mesma após realizar outras ações.

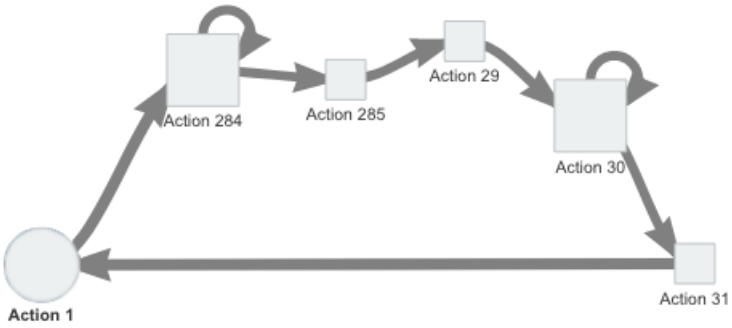


Figura 5 – Exemplo de grafo representando uma sessão de reinício. Fonte: Autor (2018).

As sessões de limiar são aquelas em que o usuário iniciou e finalizou a execução de uma tarefa, mas em algum ponto teve de interrompê-la por algum motivo não relacionado à usabilidade da aplicação (um intervalo para almoço, por exemplo). Elas são caracterizadas por possuírem um intervalo de tempo muito grande (mais especificamente, acima de um determinado limiar) entre duas ações. Assim como ocorre com as sessões completas, os usuários dessas sessões passam pelos pontos iniciais e finais, contudo, pelo uso ter sido influenciado por fatores externos, essas sessões são desconsideradas durante a busca por *smells*.

A Figura 6 mostra um exemplo de sessão de limiar. Nela, o usuário começou com a ação inicial *Action 1* e terminou com a ação final *Action 198*, contudo, o intervalo de tempo entre um dos pares de ações sequenciais[3] ultrapassou o limiar definido.

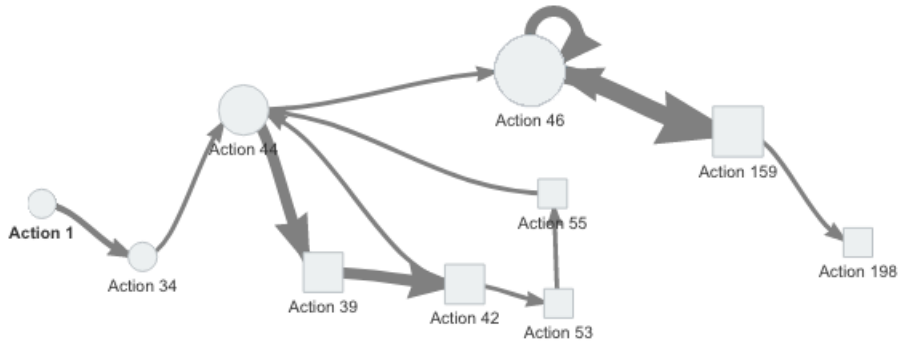


Figura 6 – Exemplo de grafo representando uma sessão de limiar. Fonte: Autor (2018).

As sessões de erro indicam os casos em que o usuário iniciou a execução de uma tarefa, mas por algum motivo não conseguiu concluí-la (ou seja, realizou uma das ações iniciais, mas nenhuma das finais). Essas sessões podem ser indicativos de problemas, uma vez que a usabilidade da aplicação pode ter sido a causa direta da adversidade do usuário. A Figura 7 mostra um exemplo de sessão de erro. Nesse caso, o usuário começou com a ação inicial *Action 1*, mas nunca chegou à nenhuma ação final.

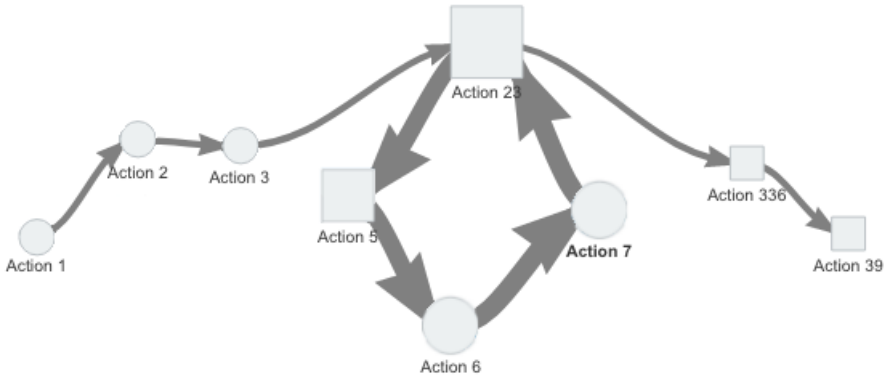


Figura 7 – Exemplo de grafo representando uma sessão de erro. Fonte: Autor (2018).

Opcionalmente, nessa etapa também é possível realizar ajustes nos parâmetros dos *usability smells*. Cada *smell* possui uma série de parâmetros que podem ser ajustados em sua estratégia de detecção. Esses parâmetros determinam a rigidez da detecção, ou seja, a quantidade de elementos da aplicação indicados como contendo *smells* varia de acordo com os valores definidos.

[3] A Figura 6 mostra apenas uma visão geral da sequência de ações realizadas na sessão, não possibilitando a identificação do ponto exato onde ocorreu a violação do limiar.

A abordagem provê uma série de valores padrão para os parâmetros de cada estratégia de detecção. Esses valores foram determinados empiricamente, escolhendo-se aqueles que geraram detecções mais precisas em diversos testes conduzidos utilizando diferentes valores para cada parâmetro. Contudo, esses testes foram baseados no conjunto de dados capturados disponíveis, referentes a um determinado sistema. Assim, eles podem não ser adequados dependendo da natureza da aplicação avaliada. Por esse motivo, os parâmetros também podem ser ajustados livremente, de acordo com as particularidades da aplicação ou o desejo do avaliador.

Por fim, as ações capturadas são analisadas por algoritmos específicos, buscando relações entre o comportamento dos usuários e a ocorrência de *smells*. Para cada *smell*, um determinado conjunto de dados (dependendo do que se deseja encontrar) dentre os *logs* capturados é utilizado, assim como, cada um deles exige um algoritmo diferente (apesar de haver pontos em comum entre alguns). Os detalhes sobre os *smells* e algoritmos utilizados são apresentados nas seção 3.2.

3.1.3 Geração de Relatórios

Na terceira etapa, os resultados das análises são exibidos. Para cada *smell*, é apresentada uma lista de ações ou sessões onde o mesmo foi detectado, as informações necessárias para identificar o local da detecção no sistema, e os *refactorings* sugeridos. Essas informações variam de acordo com o conceito de cada *smell*, e tem por objetivo não apenas mostrar o local problemático, mas também ajudar a identificar a característica do sistema que origina o problema, para que os *refactorings* possam ser aplicados corretamente.

Os relatórios gerados são os artefatos que proveem auxílio ao avaliador na detecção de problemas de usabilidade. A definição de cada *smell* indica o comportamento problemático que deve ser evitado, assim, o avaliador sempre sabe o que procurar, mesmo que tenha pouca experiência com usabilidade. Além disso, a implementação da abordagem também provê dicas mais específicas sobre alguns dos problemas comuns associados a cada *smell*, conforme será apresentado na seção 3.3.

3.2 Catálogo de *Usability Smells*

A fim de demonstrar a aplicação da abordagem descrita nesta dissertação, foi definido um conjunto de *usability smells* para ter sua detecção automática implementada. A escolha dos *smells* foi baseada na abrangência de fontes para a definição de *smells* (como uma forma de analisar a escalabilidade desse processo) e na facilidade em integrá-los à abordagem proposta. Esse catálogo conta, atualmente, com sete *smells* definidos em termos de seu conceito, estratégia de detecção e *refactorings* sugeridos.

Um resumo sobre a origem e os dados utilizados pela estratégia de detecção de cada *usability smell* é mostrado na Tabela 1. Neste caso, "Proposto" indica que o *smell* e sua estratégia de detecção foram definidos nesta dissertação; "Adaptado" indica que a estratégia de detecção do *smell* foi proposta neste trabalho, mas seu conceito vem da definição de outro trabalho; e "Replicado" indica que seu conceito e estratégia de detecção foram propostos em outro trabalho.

Tabela 1 – Origem e dados de entrada para cada *smell*.

Nome	**Origem**	**Entrada**
Laborious Task	Proposto	Sessões
Cyclic Task	Proposto	Sessões
Lonely Action	Proposto	Ações
Too Many Layers	Adaptado (ALMEIDA et al., 2015)	Sessões
Undescriptive Element	Replicado (GRIGERA et al., 2017)	Ações
Missing Feedback	Adaptado (HARMS; GRABOWSKI, 2014)	Ações
Fat Interface	Adaptado (SOUSA et al., 2017)	Ações

3.2.1 *Laborious Task*

Esse *smell* foi definido com base em um estudo experimental realizado com a ferramenta UseSkill (SOUZA et al., 2016a), através da observação das características das sessões confirmadas como indicativos de problemas de usabilidade. Ele ocorre quando é necessário realizar uma grande quantidade de ações para cumprir uma determinada tarefa. Uma tarefa com muitas ações exige um grande esforço e consome muito tempo dos usuários, o que deve ser evitado não só para aumentar a produtividade como também o engajamento dos mesmos. Além disso, a chance do usuário ser interrompido nessa tarefa é grande, uma vez que é longa, e, caso a interrupção seja demorada, a chance de perder um longo tempo de trabalho torna-se alta.

A detecção do *Laborious Task* é realizada verificando-se a quantidade de ações e o tempo despendidos em cada sessão. Se a quantidade de ações e o tempo são muito elevados para uma determinada sessão, ou seja, se esses valores ultrapassam um determinado limiar, a tarefa associada àquela sessão é considerada como contendo o *smell Laborious Task*. O valor padrão para esses limiares é determinado com base na barreira externa superior das quantidades de ações e tempo do conjunto de todas as sessões de entrada, calculada conforme a Equação 3.1. Onde F_o representa a barreira externa (*outer fence*); $Q1$ o quartil inferior; $Q3$ o quartil superior; e 3 é o valor que indica que a barreira calculada é externa.

$$F_o = Q3 + (Q3 - Q1) \times 3 \tag{3.1}$$

Os quartis são valores que dividem o conjunto de dados ordenados em quatro partes. O primeiro quartil ($Q1$) é o número que deixa 25% das ocorrências abaixo e 75% acima,

enquanto o terceiro quartil ($Q3$) é o que deixa 75% abaixo e 25% acima. A mediana ($Q2$) é o valor que deixa 50% das observações abaixo e 50% acima, porém, a mesma não é usada no cálculo das barreiras.

Qualquer valor acima da barreira superior ou abaixo da barreira inferior é considerado um *outlier*, ou seja, uma observação que está bastante afastada das demais. Para o *smell Laborious Task*, apenas a barreira superior foi considerada, pois apenas as sessões atipicamente grandes são indicativos de tarefas que demandam muito esforço do usuário.

O cálculo automático de ambos os limiares, de quantidade de ações e tempo, é realizado utilizando-se como entrada o conjunto de quantidade de ações e tempo, respectivamente, de todas as sessões completas. Optou-se por realizar o cálculo apenas com essas sessões pois, em teoria, elas representam a utilização correta da funcionalidade, reduzindo o risco de "ruídos" nos resultados. Além disso, apenas as ações explicitamente executadas pelos usuários são contadas em todos os cálculos que envolvem esse *smell*, para garantir que o excesso de ações está realmente relacionado à um grande nível de esforço.

É possível observar que o valor determinado pela equação Equação 3.1 não é adequado para a detecção se todas as sessões de entrada forem grandes, ou seja, se todos os valores forem próximos, não haverá *outliers*, mesmo que a tarefa analisada exija muitas ações. Contudo, a ideia é utilizar as sessões referentes à todas as tarefas analisadas como entrada. Assim, pode-se estabelecer automaticamente o "comportamento normal" da aplicação e apontar aqueles que destoam muito desse comportamento.

O *refactoring* sugerido para esse *smell* é simplificar a realização da tarefa tanto quanto possível, removendo-se as ações desnecessárias e auxiliando o usuário a executá-la de forma eficiente. Isso reduz o esforço necessário para utilizar a aplicação e otimiza o tempo dos usuários, consequentemente aumentando seu interesse. Outra possibilidade, caso a tarefa não possa ser reduzida, é dividi-la em etapas e armazenar os resultado das ações executadas em cada uma dessas etapas para evitar a perda de informação.

3.2.2 *Cyclic Task*

Esse *smell* também foi definido com base na observação dos problemas detectados no estudo experimental realizado com a ferramenta UseSkill (SOUZA et al., 2016a). Ele ocorre quando a tarefa executada exige a realização de muitas ações repetitivas e, assim como *Laborious Task*, é um indicativo de que a tarefa em questão pode ser exaustiva e tediosa para o usuário.

Para verificar se a sessão possui ações (ou conjuntos de ações) repetitivas é necessário analisá-la como um grafo. Nesse grafo de sessão, os vértices correspondem às ações realizadas, e as arestas, ao caminho percorrido pelo usuário. Buscar por ações repetitivas, então, corresponde a buscar por ciclos repetidos nesse grafo. Entretanto, apenas conter

ciclos repetidos não é suficiente para afirmar que uma tarefa é repetitiva, pois, dependendo do tamanho da tarefa, esses ciclos podem representar uma parte ínfima da mesma.

Assim, para determinar se uma sessão contém o *Cyclic Task*, utilizou-se uma medida definida como "taxa de ciclos". Essa taxa determina a proporção de ações cíclicas (ou repetitivas) na sessão e é calculada de acordo com a Equação 3.2. Onde C_r representa a taxa de ciclos (*cycle rate*) da sessão; A_c o número total de ações da sessão que estão contidas em ao menos um ciclo; e A_t o total de ações da sessão.

$$C_r = \frac{A_c}{A_t} \times 100 \tag{3.2}$$

Se alguma das sessões de uma tarefa possui a taxa de ciclos acima de um determinado limiar, essa tarefa é considerada como contendo o *smell Cyclic Task*. Uma boa maneira de encontrar a fonte desse *smell* é observar a ação que inicia o ciclo, pois ela indica o ponto exato onde o problema começa a acontecer.

O valor padrão para o limiar de taxa de ciclos máxima foi determinado através de testes com *logs* previamente capturados. Observou-se que as sessões com taxa de ciclos menor ou igual a 70%, ou não indicavam problemas (falsos positivos), ou indicavam problemas já presentes em sessões com taxas de ciclos maiores (redundância). Assim, esse foi o valor padrão definido para esse parâmetro.

Algumas outras filtragens também são realizadas para o caso desse *smell*. Primeiramente, para evitar a geração de conclusões incorretas, somente as sessões completas foram consideradas em todos os cálculos. A formação de ciclos também pode ocorrer por erros no sistema ou confusão do usuário (com o mesmo retornando a ação inicial após se perder na realização da tarefa, por exemplo). Apesar de esses eventos, de forma semelhante, indicarem problemas no sistema, esses casos não estão associados à realização de atividades repetitivas, sendo assim, removidos.

Além disso, as sessões com uma quantidade de ações muito pequena também foram removidas para filtrar ruídos nos dados (sessões onde o usuário realizou apenas duas ações iguais, por exemplo, poderiam ser consideradas como indicativo do *smell* sem essa filtragem). O valor padrão para esse limiar foi definido como 10, pois mesmo que sessões com tamanho menor que esse possam conter ciclos de ações repetidas, elas não representam uma atividade exaustiva devido à baixa quantidade de ações.

Por fim, o *refactoring* sugerido para esse *smell* é implementar mecanismos na aplicação que permitam a realização da tarefa de forma mais dinâmica e/ou, se possível, automatizar essas ações repetitivas. O *Cyclic Task* é constantemente causado por uma má organização das ações que devem ser executadas durante o fluxo da tarefa. Assim, pequenas alterações na interface podem resultar em um grande aumento na eficiência de execução da tarefa afetada.

3.2.3 *Lonely Action*

A ideia desse *smell* surgiu através da observação de uma aplicação Web problemática. Ele ocorre quando, em uma página Web, uma única (e mesma) ação é realizada na maior parte das vezes. Se a ação realizada é quase sempre a mesma, isso quer dizer que a página possui uma única ação ou apenas uma das ações dessa página é relevante, ou ainda que as demais ações têm pouco destaque e passam despercebidas pelos usuários.

De toda maneira, esse *smell* pode ser um sinal de que algo incomum está ocorrendo na página afetada, e isso deve ser investigado. A Figura 8 mostra um exemplo desse comportamento, onde, após realizar um cadastro de frequência de alunos, o usuário é redirecionado à uma tela cuja única opção é fechar.

Figura 8 – Exemplo de tela afetada com o *smell Lonely Action*. Fonte: Autor (2018).

O *Lonely Action* está associado a ações específicas (em vez de sessões), portanto, é necessário analisá-las individualmente. Assim, o processo de detecção se inicia pela determinação do conjunto de todas as URLs diferentes das ações capturadas. Para cada URL associa-se, então, o conjunto de todas as ações que ocorrem na mesma e a "taxa de ocorrência" de cada uma dessas ações, calculada conforme a Equação 3.3. Onde O_r representa a taxa de ocorrência (*occurrence rate*); A_n é a quantidade de ocorrências da ação n na URL; e A_t o número total de ações que ocorrem na URL;

$$O_r = \frac{A_n}{A_t} \times 100 \tag{3.3}$$

O *smell* é detectado caso a taxa de ocorrência (O_r) da ação exceda um determinado limiar. A ideia é determinar a proporção de ocorrência de uma ação em uma determinada página que indica que a mesma é realizada um número desproporcionalmente superior de vezes em comparação às demais.

O valor padrão para o limiar é calculado com base no conjunto de taxas de ocorrência de todas as ações realizadas na página. Ele é igual a cinco vezes a taxa de ocorrência da segunda ação mais frequente na URL. O valor 5 foi determinado através de testes, onde valores inferiores a esse produziram detecções sobre elementos não problemáticos (falsos positivos) e valores superiores impossibilitaram a detecção de elementos problemáticos (verdadeiros negativos).

A ideia por trás do cálculo é que, se uma ação ocorre mais que cinco vezes a segunda ação mais frequente, ela também ocorre nessa mesma proporção ou superior em relação às demais ações, o que representa uma diferença significativa. Uma característica particular desse cálculo é o fato de ele ser dinâmico, ou seja, como a comparação deve ser realizada entre ações em uma mesma URL, o cálculo é refeito para cada URL analisada.

Em adição, também é possível estabelecer um limiar de quantidade mínima de ocorrências, com o objetivo de evitar a detecção de falsos positivos. Sem esse limiar, caso sejam capturadas apenas cinco ações em uma página, por exemplo, e quatro delas sejam iguais, a detecção ocorrerá, apesar de não ser possível fazer conclusões significativas sobre esse caso, devido à baixa quantidade de ocorrências. O cálculo para o valor padrão desse limiar é feita com base na mediana das quantidades de ações do conjunto de URLs. Observou-se, através de testes, que uma grande quantidade de URLs possui um número muito pequeno de ações (não suficiente para inferir conclusões válidas), assim, a "metade maior" da amostra de dados carrega os resultados mais significativos.

Em relação aos eventos analisados, somente são considerados aqueles efetuados diretamente e intencionalmente pelo usuário. Assim, os eventos do tipo *mouseover* (posicionar o ponteiro do mouse sobre um elemento por um determinado tempo) e *onload* (carregamento de uma página) são desconsiderados dos cálculos.

Se existe uma única ação em uma página, é provável que a mesma não necessite ser executada manualmente pelo usuário, portanto, o *refactoring* sugerido é automatizá-la. Se há mais de uma ação e todas possuem importância semelhante, a sugestão é rever o *design* da interface para se assegurar que todas elas estão igualmente visíveis ao usuário. Caso as ações secundárias sejam pouco realizadas porque, de fato, têm pouca importância, as mesmas devem ser removidas, se possível, para manter o *design* minimalista (NIELSEN, 1993).

3.2.4 *Too Many Layers*

Esse *smell* foi adaptado a partir do descrito por Almeida et al. (2015). Considera-se que ele aparece quando o usuário tem de passar por muitas páginas da aplicação para conseguir realizar uma determinada tarefa. Passar por várias páginas para realizar uma tarefa afeta severamente o desempenho do usuário, pois o mesmo precisa lembrar de todos os passos que irá executar e que já foram executados.

A abordagem de Almeida et al. (2015), entretanto, é executada em ambiente *Desktop*, onde considera-se como página cada janela da aplicação. Para as aplicações Web, neste trabalho será considerado como página a cada interface associada a uma URL única. Portanto, para se saber por quantas páginas o usuário teve de passar, basta obter a quantidade de URLs diferentes na sessão. Se essa quantidade ultrapassar um dado limiar,

o *smell* é detectado para a tarefa correspondente. O objetivo aqui é determinar uma quantidade de páginas máxima aceitável, onde, após esse ponto considera-se que o excesso de páginas irá prejudicar o desempenho do usuário na execução da tarefa.

Para a determinação do valor padrão do limiar, o sistema utilizado durante a avaliação da abordagem foi analisado. Nessa análise, observou-se que, durante execução de uma tarefa, independente de qual seja ela, a quantidade de URLs varia, em média, de 3 a 5. Assim, o valor padrão para esse limiar foi definido como 5. A estratégia de detecção desse *smell* também considera apenas as sessões completas nos cálculos, para garantir que a detecção esteja relacionada à execução inerente da tarefa, e não à um erro isolado.

É possível perceber que o *Too Many Layers* é semelhante ao *Laborious Task*, uma vez que ambos podem identificar tarefas que demandam grande esforço. A diferença essencial entre os dois é que o *Too Many Layers* avalia a quantidade de URLs, enquanto o *Laborious Task* avalia a quantidade de ações. O usuário pode, por exemplo, realizar centenas de ações em uma única página, o que seria detectado pelo *Laborious Task*, mas não pelo *Too Many Layers*. Em contrapartida, ele também poderia realizar poucas ações, mas ter que passar por dezenas de páginas diferentes para isso, o que seria detectado pelo *Too Many Layers*, mas não pelo *Laborious Task*.

Uma limitação clara dessa estratégia de detecção é que somente são consideradas páginas Web diferentes aquelas com URLs diferentes. Algumas aplicações utilizam Ajax (GARRETT, 2005) para realizar mudanças na interface sem alterar a URL, consequentemente, impossibilitando a detecção pela estratégia proposta. Atualmente, a forma de captura de dados utilizada pela ferramenta UseSkill não permite a identificação desse tipo de comportamento, entretanto, caso o problema também envolva muitas ações, ele pode ser detectado pelo *Laborious Task*.

O usuário não deve ser obrigado a lembrar de todo o caminho percorrido entre as páginas, portanto, é sempre preferível que as ações sejam fáceis de reconhecer, em vez de terem que ser memorizadas (NIELSEN, 1993). Assim, para uma tarefa detectada com o *Too Many Layers*, o *refactoring* sugerido é agrupar as ações relacionadas a ela em uma mesma página, na forma de menus, por exemplo, evitando, assim, a formação de muitas camadas. Caso isso não seja viável, outra possibilidade é manter um *design* semelhante entre as páginas envolvidas para facilitar o reconhecimento e prover mecanismos que ajudem o usuário a se localizar no caminho percorrido, como barras de navegação, por exemplo.

3.2.5 *Undescriptive Element*

Conforme o descrito na abordagem de Grigera et al. (2017), esse *smell* aparece quando muitos usuários tentam conseguir uma *tooltip* de um elemento de uma página Web,

o que pode indicar que esse elemento não é autodescritivo o suficiente. O *Undescriptive Element* foi adaptado para funcionar de acordo com a abordagem proposta nesta dissertação, contudo, o seu conceito e estratégia de detecção mantiveram-se quase inalterados, uma vez que ambas as abordagens utilizam o mesmo contexto (Web). A Figura 9 mostra uma captura de tela de uma versão antiga do Google Drive que exemplifica esse comportamento. Nessa tela foi observado que os usuários constantemente procuravam pela opção de *log out* posicionando o mouse sobre seu respectivo nome (GRIGERA et al., 2017).

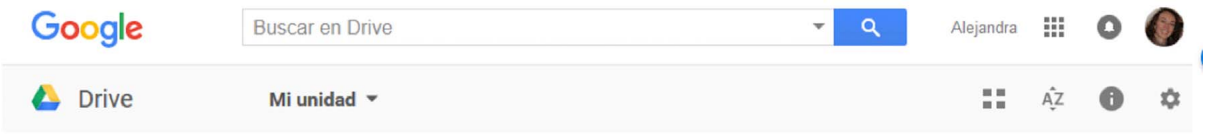


Figura 9 – Exemplo de tela afetada com o *smell Undescriptive Element*. Fonte: Grigera et al. (2017).

Considera-se que ocorreu uma "tentativa de *tooltip*" quando o usuário mantém o ponteiro do mouse parado sobre um elemento da página (*mouseover*) por um determinado tempo. O *tooltip* é uma pequena caixa de texto que aparece sobre o elemento, contendo uma explicação adicional sobre o mesmo. Geralmente, o usuário busca por essa explicação quando não entendeu o que aquele elemento representa. Assim, se há um grande número dessas tentativas em um mesmo elemento, isso pode ser um sinal de que o significado desse elemento não está claro para os usuários.

Para a detecção, conta-se o número de ocorrências do evento *mouseover* de todos os elementos, em um determinado intervalo de tempo. Se o número de ocorrências nesse intervalo exceder um dado limiar, o *smell* é detectado no elemento em questão. Apenas os elementos do tipo *TextField* (campo de texto) foram descartados durante essa análise, pois é comum que os usuários deixem o ponteiro parado sobre o mesmo após clicá-lo para realizar o preenchimento.

O trabalho de Grigera et al. (2017) define o limiar padrão para a estratégia de detecção desse *smell* como 30 tentativas em um intervalo de 5 dias. Além disso, ela também define o *refactoring* para o mesmo, que consiste em renomear o elemento afetado e/ou mudar seu *widget*. Ou seja, caso o elemento seja apenas textual, seu texto deverá ser alterado para representar melhor a sua função, e, caso contenha outros elementos (figuras, por exemplo), os mesmos também devem ser revistos e, possivelmente, alterados.

3.2.6 *Missing Feedback*

Esse *smell* foi originalmente proposto por Harms e Grabowski (2014), e ocorre quando uma determinada ação é constantemente repetida. Os usuários tendem a repetir uma ação quando não têm certeza se ela foi processada pelo sistema, logo, esse é um sinal de que não há *feedback* relacionado à ação em questão. A falta de *feedback* causa confusão

no usuário e a repetição de ações também pode ocasionar erros no processamento do sistema.

A abordagem de Harms e Grabowski (2014) utiliza a busca por iterações em árvores de tarefas para identificar o *Missing Feedback*. Contudo, essa estratégia de detecção não pode ser utilizada pela abordagem desta dissertação, assim, uma nova estratégia teve de ser proposta. Neste trabalho a detecção é determinada pela mediana da quantidade de repetições da ação.

Primeiramente, são mapeadas todas as instâncias de ocorrências de cada ação que foi realizada ao menos duas vezes seguidas. A cada instância de cada ação mapeada é associado a sua respectiva quantidade de repetições, e a mediana é calculada a partir do conjunto de instâncias de cada ação. Por exemplo, se um usuário A executa uma ação X uma vez, um usuário B executa a ação X duas vezes seguidas e um usuário C executa a ação X, em determinado momento 4 vezes seguidas, e em outro momento 5 vezes seguidas, temos 4 instâncias da ação X com mediana de repetições igual a 3.

Optou-se por utilizar a mediana, em vez de instâncias individuais, para garantir que o comportamento detectado ocorre com frequência e não é resultado de um problema isolado. Geralmente, realizar uma ação duas ou três vezes seguidas (como um clique em determinado elemento, por exemplo) já é um sinal de que usuário está em dúvida se sua ação teve efeito. Contudo, isso também pode ser resultado de ações acidentais. Assim, para garantir que a detecção não ocorra nesse tipo de situação, o valor padrão para a quantidade máxima de repetições foi definido como 3.

Adicionalmente, também é possível definir uma quantidade mínima de instâncias da ação, para evitar que a detecção ocorra em casos onde a quantidade de instâncias é muito pequena para gerar conclusões válidas. O valor padrão para esse limiar foi definido como 5, pois através de testes observou-se que valores inferiores à esse podem gerar detecções de falsos positivos.

A abordagem de Harms e Grabowski (2014) não sugere nenhum *refactoring* para esse *smell*. Contudo, como ele representa um comportamento cuja causa é bem específica, pode-se definir seu *refactoring* de maneira bem direta como sendo indicar ao usuário que a ação foi processada pelo sistema. Esse *smell* é bem comum em componentes que realizam algum tipo de carregamento de dados assíncrono. Assim, existem diversos *widgets* que podem ser usados para dar essa indicação ao usuário, como barras e ícones de carregamento, por exemplo.

3.2.7 *Fat Interface*

Esse *smell* foi parcialmente baseado no problema de *design Fat Interface* (MARTIN; MARTIN, 2006; SOUSA et al., 2017) e na heurística "estética e design minimalista" (NIEL-

SEN, 1993). Ele ocorre quando a interface da aplicação possui muitos elementos interativos[4] diferentes. Quanto mais informações estiverem na tela, mais o usuário terá que gastar tempo lendo e tomando decisões, por isso é importante manter apenas as opções essenciais. Um exemplo de tela de uma aplicação Web afetada com o *smell Fat Interface* é mostrado na Figura 10.

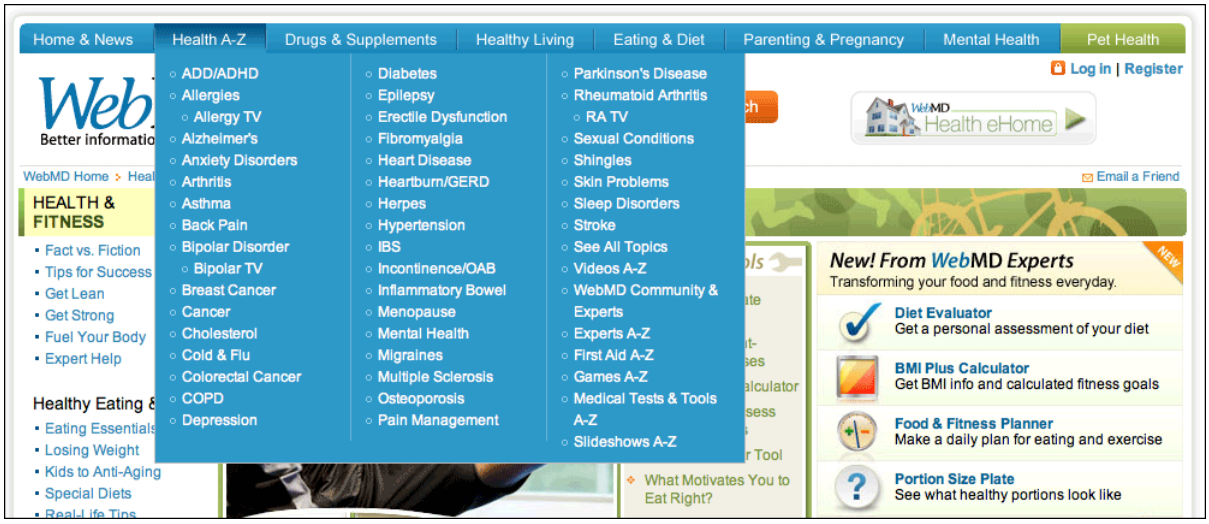


Figura 10 – Exemplo de tela afetada com o *smell Fat Interface*. Fonte: Tidwell (2010).

A estratégia de detecção proposta nesta dissertação para o *Fat Interface* é avaliar a quantidade de ações diferentes que ocorrem em cada página Web. Analogamente ao que ocorre com o *smell Too Many Layers*, nesta estratégia considera-se como página a cada URL diferente entre os dados capturados. Para cada URL é, então, associado o conjunto de todas as ações diferentes que ocorrem na mesma, e, se a quantidade dessas ações ultrapassar um determinado limiar, a página em questão é apontada como contendo esse *smell*.

Uma particularidade da estratégia de detecção desse *smell* é que todos os elementos interagidos são contados, assim, ele também pode ser detectado em páginas onde os usuários tentam interagir com elementos que não fazem parte dos menus da aplicação. Optou-se por utilizar essa estratégia pois assim é possível avaliar não apenas a interface em si, mas também a percepção do usuário sobre o que faz parte dela. Uma ferramenta de análise de código fonte pode facilmente determinar a quantidade de elementos interativos de uma página Web (ALONSO-RíOS et al., 2009), mas somente através da análise das ações realizadas pelos usuários é possível determinar quais elementos são percebidos por eles como interativos.

[4] Nesta dissertação, é chamado de elemento interativo todo aquele sobre o qual a ação do usuário provoca alguma alteração no sistema (como botões, campos de texto, etc.) e de elemento interagido todo aquele sobre o qual o usuário executou alguma ação, independente de ele ser interativo ou não (como um clique no plano de fundo da aplicação, por exemplo).

O limiar para a quantidade de ações diferentes máxima também foi definido através de testes empíricos. Considerando o conjunto de dados com o qual a estratégia de detecção foi testada, observou-se que é bem comum que as interfaces possuam até 30 elementos interagidos diferentes, então esse foi o valor padrão definido para esse limiar. Além disso, os eventos de carregamento de página e *mouseover* são descartados dessa análise, pois esses constantemente ocorrem independentemente da vontade do usuário.

O *refactoring* sugerido para esse *smell* é reestruturar a interface, removendo-se os elementos redundantes e/ou não utilizados (caso existam) e realocando-se, na hierarquia de páginas da aplicação, as opções menos utilizadas. Os elementos pseudointerativos também precisam ser redesenhados para explicitarem melhor sua função no sistema, garantindo que a interface apresentada ao usuário seja simples e de fácil entendimento.

3.3 Implementação

A ferramenta UseSkill (SOUZA et al., 2016a) foi utilizada como parte da abordagem proposta nesta dissertação, sendo responsável pela captura e parte da análise de dados. A captura de ações ocorre por meio de um componente desenvolvido em *JavaScript* que interage com o navegador registrando diversas informações sobre cada ação executada pelo usuário (como tipo, local, horário, etc.).

Para a definição das tarefas, a UseSkill também provê um *plugin* de auxílio, que identifica instantaneamente qualquer ação executada no navegador. Assim, o avaliador pode simplesmente executar as ações iniciais e finais da tarefa e copiar o identificador gerado. Por fim, essas definições são utilizadas para agrupar as ações capturadas na forma de sessões de usuário.

A detecção automática de *usability smells* foi implementada e integrada como uma extensão da ferramenta UseSkill, desenvolvida em linguagem Java[5]. A implementação consiste em uma série de algoritmos, cada um desenhado para detectar um *smell* específico, que recebem como entrada os dados capturados e agrupados pela UseSkill e apresentam como saída a lista de elementos detectados com *smells*. Junto de cada um deles, também são providas as informações necessárias para identificar esses elementos e os *refactorings*, de acordo com os *smells* presentes.

O *front-end* foi desenvolvido nas linguagens HTML, CSS, e *JavaScript*, com auxílio do *framework* Angular[6] e da biblioteca NVD3[7], que foram de importante auxílio na apresentação dos gráficos e grafos gerados pela abordagem. Todas essas linguagens utilizadas já faziam parte do projeto original da ferramenta, assim, elas foram escolhidas para facilitar

[5] https://www.java.com/
[6] https://angularjs.org
[7] http://nvd3.org/

a integração ao seu código fonte.

No *back-end*, O SGBD (Sistema de Gerenciamento de Banco de Dados) MySQL[8] foi usado para armazenar os dados resultantes da captura de dados realizada pela ferramenta UseSkill, além das configurações particulares de cada teste criado na mesma (como os valores definidos para os parâmetros do *smells*, por exemplo). O *framework* Hibernate[9] foi utilizado para facilitar o mapeamento dos objetos entre as classes Java e o Banco de Dados. O Maven[10] ficou com a responsabilidade de gerenciar essa e outras dependências, como as bibliotecas JGraphT[11] e Google Guava[12]. A arquitetura das tecnologias utilizadas nesta implementação é apresentada na Figura 11. A arquitetura completa com todas as tecnologias utilizadas pela UseSkill poder ser vista em Souza (2016).

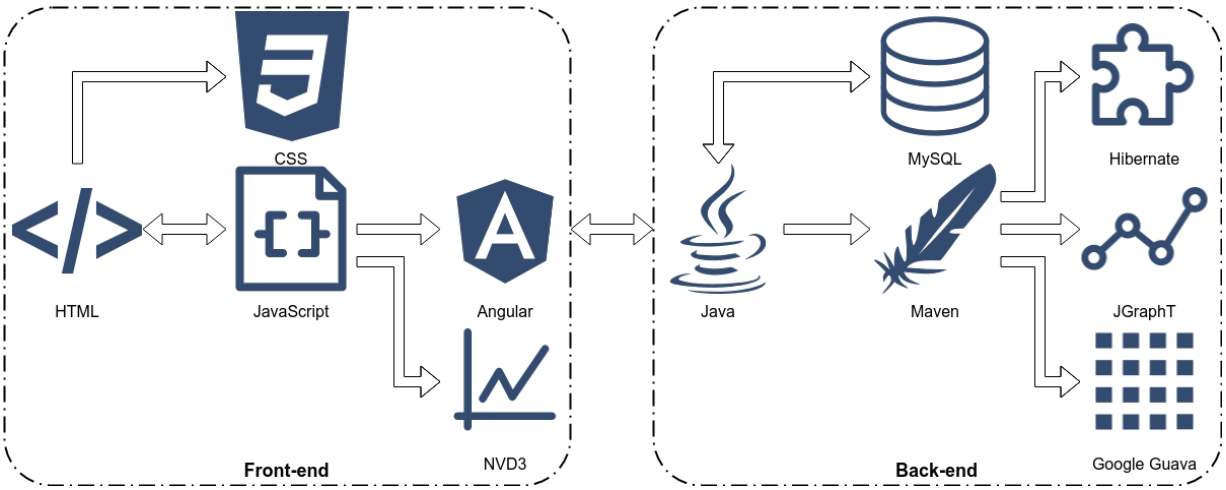


Figura 11 – Arquitetura de tecnologias utilizadas na implementação da extensão da UseSkill. Fonte: Autor (2018).

A JGraphT assistiu na manipulação de grafos, provendo estruturas de dados simplificadas e algoritmos para trabalhar com essas estruturas e a Google Guava forneceu algumas estruturas de dados auxiliares (como os *Multisets*), que simplificaram e otimizaram a realização das análises. A implementação também foi dividida em dois módulos: "estatísticas do sistema" "e detecção de *usability smells*".

3.3.1 Módulo de Estatísticas do Sistema

O módulo de estatísticas apresenta gráficos sobre o estado do sistema em relação às principais métricas usadas como parâmetros para a detecção de *smells*. Para os *smells* associados a tarefas, é apresentado o gráfico de distribuição das sessões. Nesse gráfico, as sessões referentes à tarefa analisada são ordenadas em função da métrica avaliada e têm

[8] https://www.mysql.com/
[9] http://hibernate.org/
[10] https://maven.apache.org/
[11] http://jgrapht.org/
[12] https://github.com/google/guava

um valor atribuído de acordo com o seu posicionamento em relação ao conjunto total de amostras (à sessão que divide o conjunto de dados ao meio, por exemplo, será atribuído o valor 50%).

A Figura 12a mostra um exemplo de gráfico gerado para a métrica "quantidade de ações" relacionando o estado de 4 tarefas. O eixo das abscissas representa o posicionamento da sessão e, o das ordenadas, sua quantidade de ações correspondente. Também é possível visualizar as sessões relacionando a quantidade literal, em vez da proporção, com a respectiva métrica. Contudo, como cada tarefa possui uma quantidade diferente de sessões capturadas, essa visualização é menos adequada para efeito comparativo. Um exemplo dessa forma alternativa de visualização, envolvendo as mesmas 4 tarefas do exemplo anterior, é mostrado na Figura 12b.

Para os *smells* associados à ações, é apresentado o gráfico das ações mais afetadas pela métrica avaliada. A Figura 13 mostra um exemplo desse tipo de gráfico, relacionando as 10 ações com os maiores valores da métrica "taxa de ocorrência" (associada ao *smell Lonely Action*), destacando os detalhes da sexta ação mais afetada.

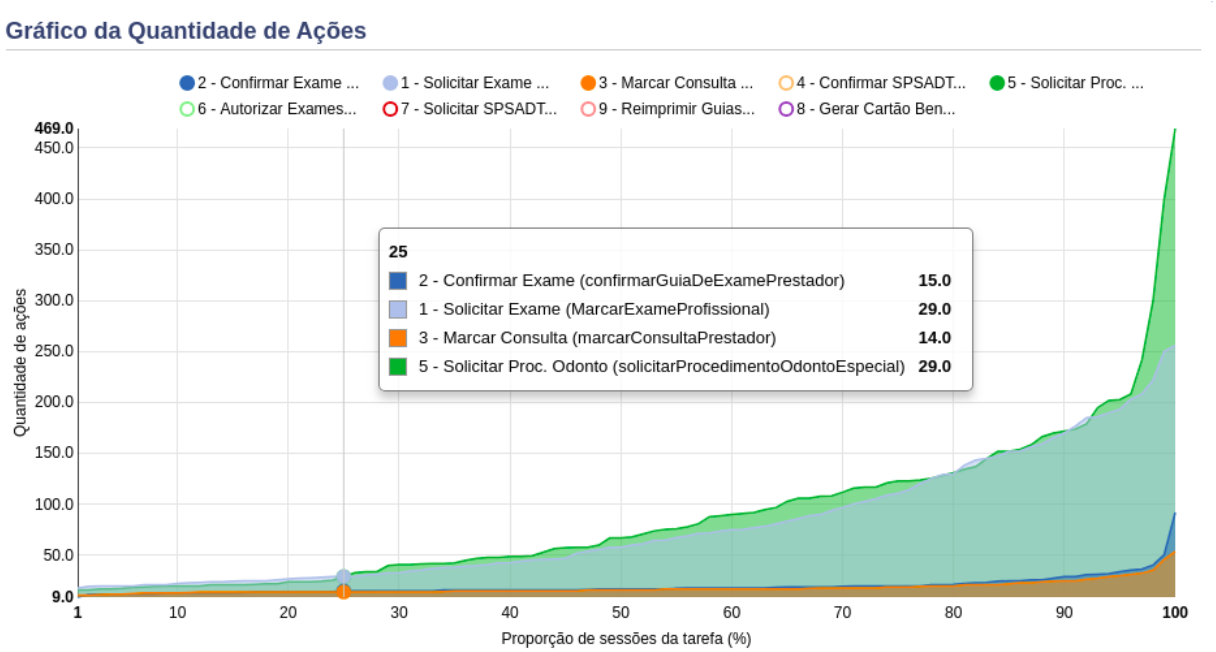
A Tabela 2 mostra todas as métricas que podem ser visualizadas através do módulo de estatísticas, bem como o *smell* a qual cada uma está associada. Esse módulo foi implementado não apenas com o objetivo de fornecer uma visão geral do sistema com relação à presença de *usability smells*, mas também como um auxílio na definição manual dos limiares dos *smells* relacionados no módulo de detecção de *smells*.

Tabela 2 – Métricas que podem ser avaliadas no módulo de estatísticas.

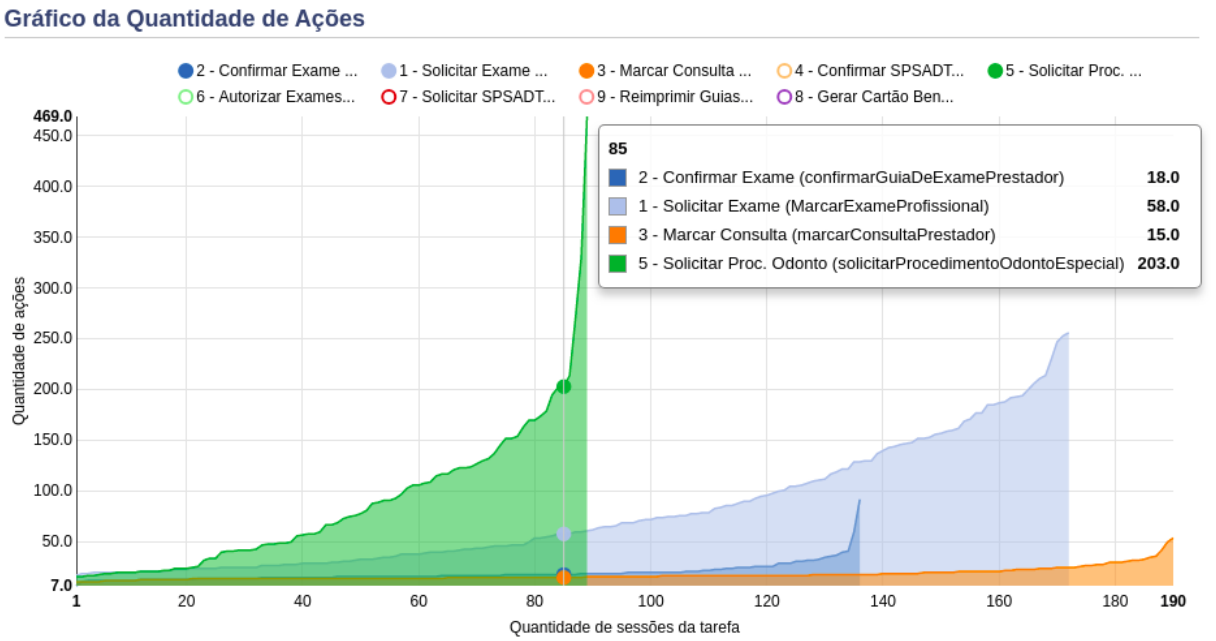
Métrica	***Smell* associado**
Quantidade de ações	*Laborious Task*
Duração (min)	
Taxa de ciclos (%)	*Cyclic Task*
Taxa de ocorrência (%)	*Lonely Action*
Quantidade de camadas	*Too Many Layers*
Quantidade de tentativas de *tooltip*	*Undescriptive Element*
Quantidade de repetições	*Missing Feedback*
Quantidade de elementos diferentes	*Fat Interface*

3.3.2 Módulo de Detecção de *Usability Smells*

O módulo de detecção de *usability smells* permite realizar a busca por *smells* automaticamente, de acordo com as configurações de parâmetros definidas. Além disso, ele também permite personalizar essa busca, escolhendo os *smells* e tarefas que o avaliador deseja que façam parte de uma análise específica. Apesar de também ser possível incluir na detecção todos os *smells* e tarefas ao mesmo tempo, essa opção dá flexibilidade ao avaliador, evitando o excesso de informações na interface quando ele deseja analisar apenas



(a) Proporção da sessão × quantidade de ações, destacando os valores na marca de 25%.



(b) Quantidade de sessões × quantidade de ações, destacando os valores na marca de 85 ações.

Figura 12 – Exemplos de gráficos relacionando as sessões com a métrica quantidade de ações para 4 tarefas. Fonte: Autor (2018).

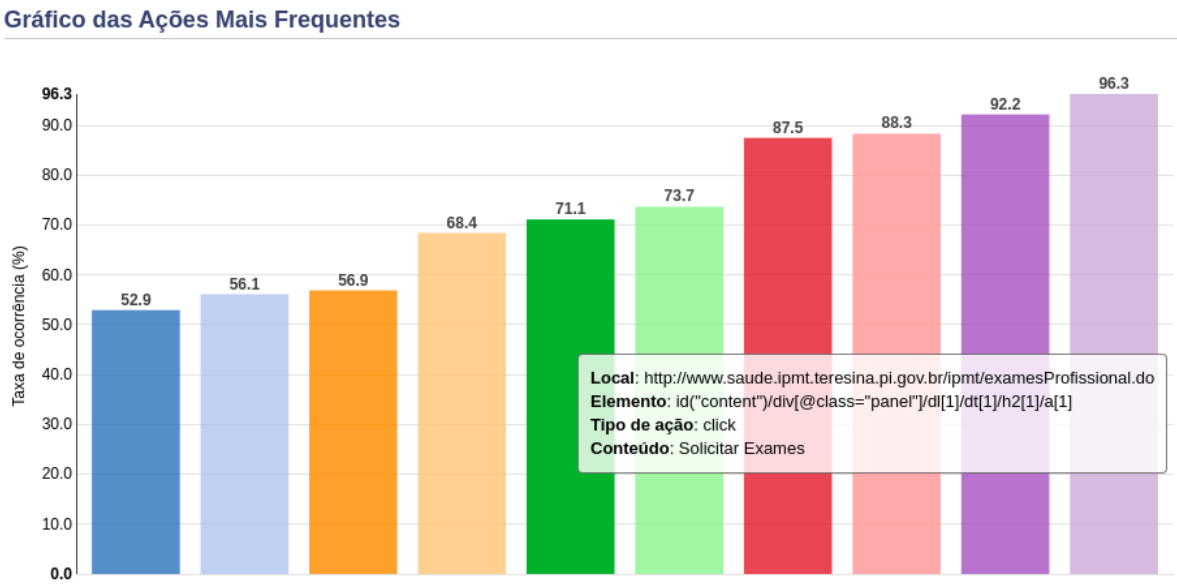


Figura 13 – Exemplo de gráfico de ações. Fonte: Autor (2018).

casos mais específicos e, possivelmente, economiza tempo, já que a duração da atividade de detecção depende da quantidade de tarefas e *smells* incluídos na mesma.

Após realizar todas as personalizações desejadas, o único requisito adicional para que se possa iniciar a detecção é definir o intervalo de tempo da mesma. Essa definição diz respeito à data em que a captura de dados ocorreu. As aplicações Web modernas estão constantemente sendo melhoradas para atender às necessidades do cliente. Assim, caso o avaliador deseje analisar uma versão específica do sistema, por exemplo, ele pode definir apenas o intervalo correspondente à essa versão.

Em seguida, os relatórios da detecção são apresentados ao avaliador, de acordo com o intervalo de tempo definido. Para cada um dos *smells* detectados na aplicação, são apresentados, inicialmente, os *refactorings* sugeridos. Além disso, algumas dicas sobre problemas mais frequentes associados ao *smell* em questão também são apresentadas como auxílio ao avaliador. A Figura 14 mostra um exemplo de tela com os *refactorings* e problemas frequentes para o *smell Lonely Action*.

Após os *refactorings*, os tipos de relatórios apresentados dependem dos dados analisados pela estratégia de detecção de cada *smell*. Para os *smells* associados à tarefas, são apresentadas as sessões onde o *smell* foi detectado, na forma de grafos. Nesses grafos interativos é possível visualizar toda a sequência de ações realizadas pelo usuário, bem como a URL, XPath, tipo e quantidade de ocorrências de cada uma dessas ações. A Figura 15 apresenta um exemplo de grafo de sessão detectado com o *smell Laborious Task*, com a ação *Action* 4 selecionada.

As sessões também são agrupadas, de acordo com a tarefa a que pertencem, e ordenadas pelo valor de seus atributos que determinam a ocorrência de *smells* (duração,

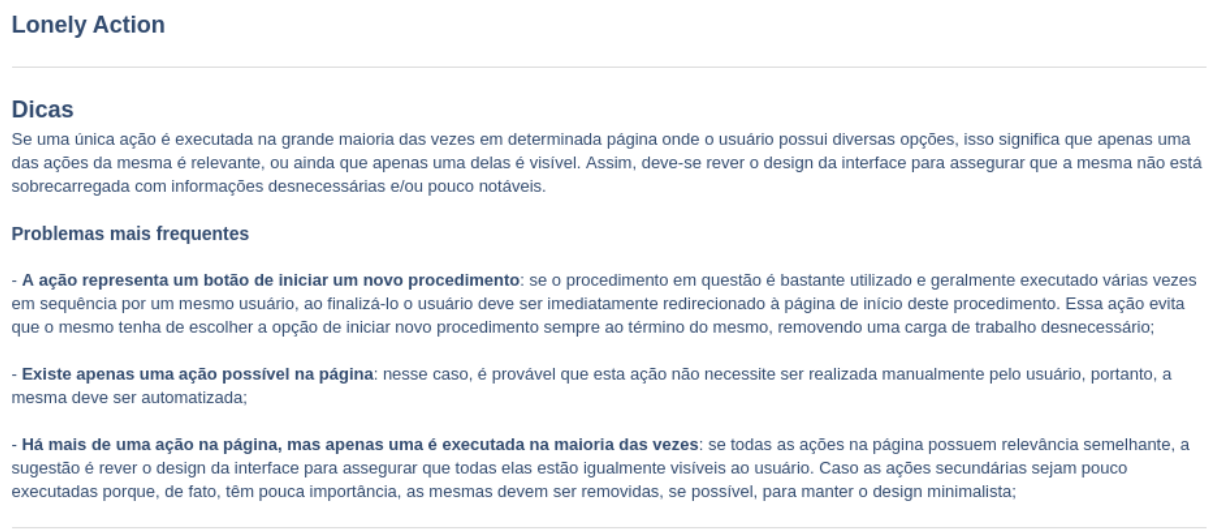


Figura 14 – Exemplo de tela de *refactorings* sugeridos para um *smell*. Fonte: Autor (2018).

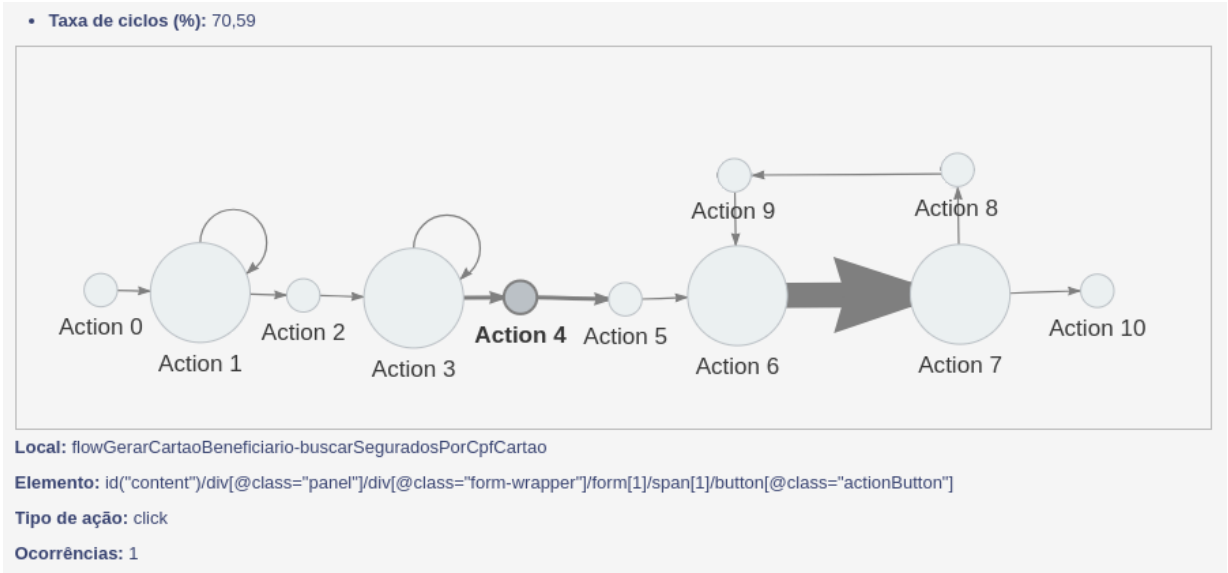


Figura 15 – Exemplo de grafo de sessão. Fonte: Autor (2018).

taxa de ciclos, etc.) do maior para o menor. Assim, o avaliador pode economizar tempo analisando os "casos mais graves" (onde há maior probabilidade de problemas) primeiro. A proporção em que a tarefa foi afetada, determinada pela divisão entre a quantidade de sessões onde o *smell* está presente e o total de sessões analisadas para a tarefa, também é apresentada.

Antes de iniciar a detecção, também é possível definir a opção de agrupar sessões similares. Se essa opção for escolhida, as sessões apresentadas na detecção são agrupadas de acordo com o nível de semelhança predefinido. O objetivo do agrupamento é diminuir as redundâncias na detecção. Quando uma determinada tarefa é muito afetada por um *smell*, a quantidade de sessões apresentadas na detecção que indicam o mesmo problema (enfrentado por diferentes usuários) é potencialmente grande. Assim, a opção de agrupamento pode

ser utilizada para economizar o tempo do avaliador.

A similaridade entre duas sessões, na forma de grafos, é determinada de acordo com o Algoritmo 1. Onde D representa a diferença entre os grafos; A é o conjunto de todas as arestas diferentes, considerando apenas a de maior peso, em caso de repetição; T é o total da soma dos pesos de todas as arestas em A; e S é a proporção de similaridade entre os grafos de entrada, numa escala de 0 a 1.

Algoritmo 1: Similaridade entre dois grafos de sessão. Fonte: Autor (2018)

Entrada: G_1, G_2
Saída: Similaridade
início
  $D = 0$
  $A = G_1.arestas \cup G_2.arestas$
  $T = \sum_{i=A(1).peso}^{A(n).peso}$
  para *cada* $a \in G_1.arestas$ **faça**
    se $a \in G_2.arestas$ **então**
      $D = D + |G_1.arestas(a).peso - G_2.arestas(a).peso|$
    senão
      $D = D + G_1.arestas(a).peso$
    fim
  fim
  para *cada* $a \in G_2.arestas$ **faça**
    se $a \notin G_1.arestas$ **então**
      $D = D + G_2.arestas(a).peso$
    fim
  fim
  $S = \frac{T-D}{T}$
fim
retorna S

Finalmente, para os *smells* associados à ações, são apresentadas as informações de identificação da ação (URL, XPath e tipo), além das informações relativas ao *smell* detectado (taxa de ocorrência, quantidade de *tooltips*, etc.). A Figura 16 mostra um exemplo de tela com as ações detectadas com o *smell Lonely Action*.

3.3.3 Preocupações com a Privacidade

Considerando que a ferramenta desenvolvida captura a interação de usuários reais com um sistema em produção, algumas medidas foram adotadas para garantir a privacidade desses usuários. A primeira delas foi se assegurar de que os dados capturados sejam completamente anônimos.

Apesar da identificação do usuário ser necessária para agrupar corretamente as ações capturadas, na forma de sessões, ela é feita através do seu ID de *login*, e não de

Figura 16 – Exemplo de tela de ações detectadas com um *smell*. Fonte: Autor (2018).

seu nome verdadeiro. Em adição, o ID de *login* do usuário também é criptografado, para evitar que seja possível identificá-lo mesmo que seu nome e ID sejam semelhantes.

Segundo, nenhuma informação preenchida em campos de texto pode ser visualizada através da ferramenta. Alguns formulários exigem o preenchimento de informações pessoais dos usuários, como telefone e endereço, assim, a visualização do conteúdo de campos de texto foi desabilitada para evitar a identificação desse tipo de informação.

Terceiro, as informações armazenadas no banco de dados da ferramenta só podem ser utilizadas com o único propósito de servirem como fonte para realização de avaliações de usabilidade. Elas não podem ser acessadas através da UseSkill de nenhuma outra forma, nem por qualquer outra ferramenta.

Por fim, a forma de captura utilizada requer acesso ao código fonte do sistema, ou seja, ela só pode ser implementada com autorização do responsável pelo mesmo. Além disso, o código fonte da implementação do *script* de captura está disponível em domínio público, o que significa que ele pode ser verificado por qualquer desenvolvedor.

3.4 Utilização

Para ilustrar como a ferramenta pode ser utilizada, será apresentado nesta seção um exemplo fictício passo a passo. Suponha que o desenvolvedor A deseja corrigir os problemas de usabilidade do seu sistema, chamado de Aplicação X. Para fazer isso com auxílio da abordagem proposta, o primeiro passo que ele deve executar é o ajuste do *script* de captura, onde o mesmo deverá, basicamente, alterar o valor (ou a fonte dele) de algumas variáveis que caraterizam o sistema, como URL, ID do usuário logado, ID do sistema, etc.

Em seguida, o desenvolvedor terá de identificar, no código fonte de seu sistema, o *template* (modelo) que é utilizado em todas as suas páginas e adicionar ao seu cabeçalho as linhas de código correspondentes ao *script* personalizado. A partir do momento em

que esse código estiver no sistema em produção, os dados de interação serão enviados diretamente para o servidor da UseSkill, toda vez que for realizado um carregamento de página no navegador do usuário.

Passado um certo período de tempo com a captura em vigor[13], o sistema estará apto para ser analisado. Isso é feito através da aplicação Web da UseSkill[14] (Figura 17). Para acessá-la é necessário que o desenvolvedor/avaliador realize o *login* (e seu cadastro, caso não possua ainda) em sua conta na UseSkill e escolha a opção "Criar Teste", no módulo *On-The-Fly*. Para criar um novo teste é preciso fornecer apenas um "Título" para o mesmo, e a URL e ID do sistema, conforme definidos no *script* de captura. Assim, os dados capturados são automaticamente associados ao teste criado, pelo ID do sistema definido.

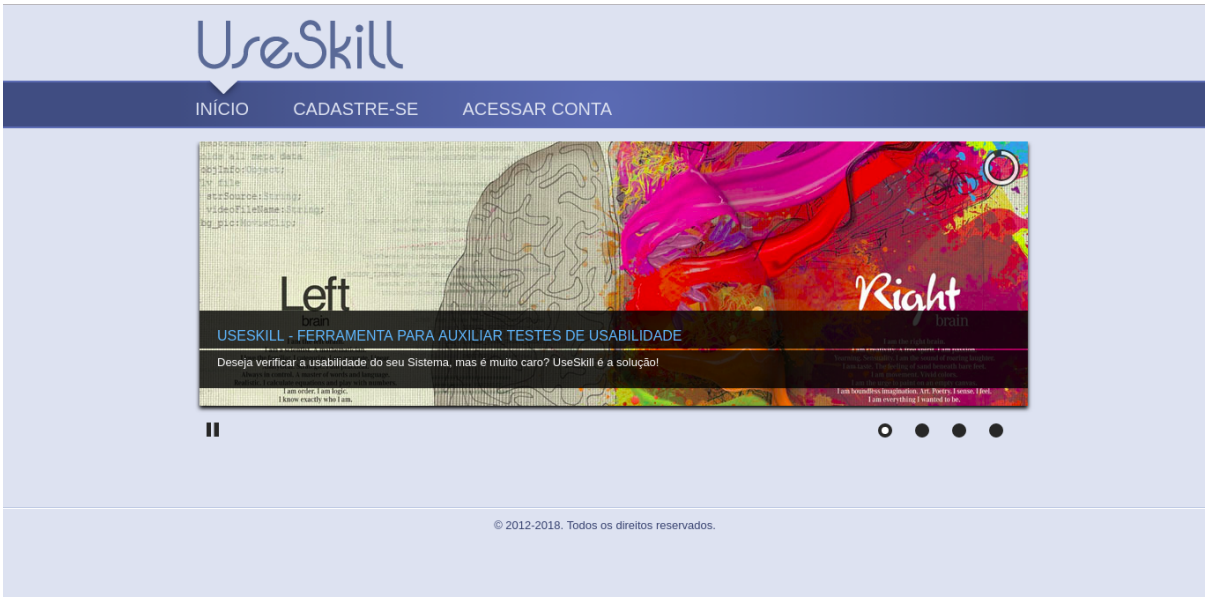


Figura 17 – Tela inicial da UseSkill. Fonte: Autor(2018).

Acessando o novo teste criado, o avaliador tem as opções de "Análise de Funcionalidades", "Estatísticas do Sistema" e "Detecção de *Usability Smells*". Dentro da parte de análise, ele deve utilizar a opção de "Gestão das Funcionalidades do Teste" para cadastrar as funcionalidades do sistema que deseja analisar. Para criar uma nova funcionalidade só é obrigatório informar um "Título" e "Tempo Máximo de Sessão"[15] da mesma e, após criada, é possível adicionar seu conjunto de ações correspondente. O processo de adicionar uma ação, por sua vez consiste em definir o seu momento (inicial ou final), tipo (clique, *mouseover*, etc.), descrição, XPath e URL.

[13] O período de captura necessário varia de acordo com a quantidade de usuários e a frequência de utilização da aplicação. Quanto maior a frequência e quantidade de usuários, menor é o tempo requerido.

[14] Atualmente disponível em <https://easii.ufpi.br/useskill/>. Contudo, dependendo da data de acesso, a extensão da nova abordagem pode ainda não ter sido integrada com o sistema em produção.

[15] Define a classificação da sessão como sendo do tipo "limiar", caso ultrapassado.

Com todas as tarefas cadastradas, o avaliador pode seguir para a detecção de *smells*. Opcionalmente, na parte de estatísticas, ele também pode informar o intervalo de tempo da detecção (Figura 18) e descobrir as partes do sistema mais afetadas por *smells* e a proporção desse impacto, para auxiliar no ajuste de parâmetros antes da detecção.

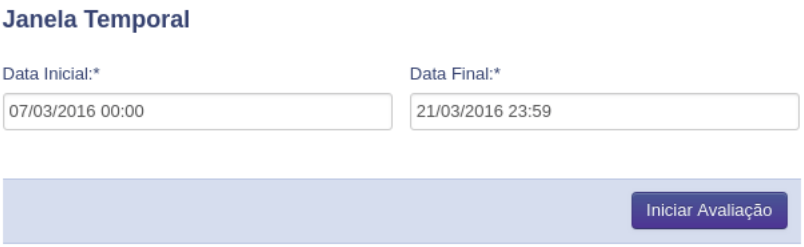


Figura 18 – Interface de definição da janela temporal da detecção. Fonte: Autor (2018).

Na parte de detecção, é possível escolher as tarefas (de acordo com as que foram definidas na etapa anterior) que serão analisadas (Figura 19b) e os *smells* que serão buscados (Figura 19a), bem como configurar os parâmetros associados a cada um deles. Por padrão, a UseSkill já vem com todos os *smells* e tarefas marcados para análise, além dos parâmetros todos definidos para seu valor padrão. Assim, caso deseje iniciar imediatamente, o avaliador precisará definir apenas o intervalo de tempo da detecção.

(a) Interface de seleção de *usability smells*.

(b) Exemplo de interface de seleção de tarefas.

Figura 19 – Telas de ajuste da detecção de *smells*. Fonte: Autor (2018).

A etapa de detecção requer um intenso processamento de dados, logo, ela pode demorar alguns minutos, dependendo da quantidade de tarefas e *smells* incluídos. Contudo, uma vez finalizada essa etapa, todos os relatórios com os resultados da busca de *smells* estão prontos para serem utilizados pelo avaliador. Assim, por fim, o avaliador irá analisar as informações fornecidas para cada *smell* detectado, constatar se os locais do sistema apontados contêm problemas de usabilidade ou não e, em caso positivo, utilizará as sugestões de *refactorings* para ajudar a corrigi-los.

3.5 Desafios e Limitações

É importante destacar algumas das limitações da abordagem proposta nesta dissertação. A primeira é referente ao contexto. Somente é possível analisar as ações que podem ser capturadas pela ferramenta UseSkill, ou seja, as ações de aplicações no contexto Web *Desktop*. Além disso, a quantidade de dados analisados também é uma limitação, ou seja, antes de realizar qualquer análise é necessário capturar um grande número de sessões de usuários (a análise também pode ser realizada com poucos dados, porém, nesse caso, a validade dos resultados pode ser comprometida). Entretanto, o contexto foi escolhido por ser bastante difundido e abranger um grande número de aplicações comerciais. Isso garante uma boa cobertura e permite gerar grandes quantidades de dados em pouco tempo, uma vez que muitas aplicações Web são utilizadas por um grande número de usuários diariamente.

Outra limitação referente a quantidade de dados é quando se tem uma quantidade muito grande para análise. A abordagem precisa realizar alguns tipos de processamento que podem demandar muito tempo e recursos computacionais quando aplicados a grandes conjuntos de dados. Para amenizar esse problema, busca-se sempre realizar esses processamentos da forma mais otimizada possível, além disso, também é possível realizar a análise sobre uma janela de tempo específica, ou seja, analisar somente o que foi coletado em um determinado intervalo de tempo.

A necessidade de alterar a aplicação testada também é uma limitação. Nem sempre é fácil ter acesso ao código fonte de aplicações em produção, principalmente de aplicações maiores, onde qualquer alteração impacta um grande número de usuários. Entretanto, a alteração que precisa ser realizada para a inclusão do *script* de captura no sistema é bem simples, e não tem grande impacto no seu funcionamento.

Além disso, a definição de tarefas pode ser um fator complicador. Determinar as ações iniciais e finais corretamente nem sempre é uma atividade trivial, e essa determinação impacta diretamente nos resultados. Para tentar amenizar esse fator, pode-se utilizar o *plugin* provido pela ferramenta UseSkill, que determina a identificação de cada ação do sistema quando a mesma é executada.

O conhecimento do avaliador em relação ao sistema também é um fator muito importante. É necessário que o mesmo conheça as partes da aplicação afetadas com *usability smells* para poder identificar corretamente as causas do problema. Isso acontece por que os *usability smells* apenas detectam sintomas relacionados a problemas. Os problemas devem ser descobertas a partir de uma pesquisa no contexto associado, feito por pessoas com conhecimento nesse contexto. Assim, em sistemas grandes, o ideal é que os relatórios da ferramenta sejam analisados por múltiplos desenvolvedores, para garantir uma maior cobertura.

Por fim, devido à captura ser realizada em ambiente não controlado, há sempre a possibilidade de haver ruídos nos dados, causados por fatores externos à usabilidade interferindo no desempenho do usuário. Contudo, conforme apresentado neste capítulo, a ferramenta que implementa a abordagem proposta provê alguns mecanismos para tentar minimizar esses ruídos. Além disso, como as análises são realizadas baseando-se no estado do conjunto, em vez de amostras específicas, a interferência causada é mínima.

3.6 Considerações Finais

Este capítulo apresentou a abordagem proposta, dividindo-a em etapas e detalhando as atividades executadas em cada uma delas. Em seguida, foi realizada a definição e apresentação dos *usability smells* que tiveram a detecção implementada nesta abordagem, de acordo com a origem e estratégia de detecção utilizada para cada um deles.

Os *smells Laborious Task*, *Cyclic Task* e *Lonely Action* foram definidos com base em estudos experimentais anteriores realizados com a ferramenta UseSkill (SOUZA et al., 2016a) e na experiência do autor com avaliações de usabilidade e sistemas com problemas de usabilidade. Os *smells Too Many Layers*, *Undescriptive Element* e *Missing Feedback* foram adaptados a partir dos trabalhos de Almeida et al. (2015), Grigera et al. (2017) e Harms e Grabowski (2014), respectivamente, demonstrando que a abordagem proposta também é capaz de integrar *smells* previamente implementados por outras abordagens. E o *smell Fat Interface* foi proposto com base em princípios de usabilidade presentes em outros trabalhos, confirmando que essa também é uma opção viável de integração.

Os *smells* definidos por Paternò, Schiavone e Conte (2017) não foram adaptados por serem mais adequados para o contexto Web móvel, como pode ser visto por sua definição. Atualmente, a abordagem proposta não é capaz de realizar a captura de eventos gerados nesse tipo de ambiente, nem leva em consideração as diferenças no comportamento do usuário no mesmo em suas análises. Entretanto, esses *smells* podem vir a ser reavaliados e integrados à ferramenta em trabalhos futuros. Os *smells* encontrados por Damevski et al. (2017) também não foram adaptados, nesse caso, por serem específicos de IDEs, não possuindo correspondência no domínio Web.

Por fim, a implementação da abordagem, aprofundando as tecnologias e detalhes técnicos, e suas limitações foram descritas. No próximo capítulo são apresentadas as avaliações da proposta e da ferramenta apresentadas neste capítulo. As avaliações foram realizadas com aplicações reais contendo centenas de usuários, o que dá uma ideia geral sobre as capacidades da ferramenta.

4 Avaliação

Este capítulo apresenta as avaliações realizadas a fim de verificar a eficácia, eficiência e viabilidade da Extensão da UseSkill com a Abordagem para Detecção de *Usability Smells* (EUSADUS). Essas as avaliações são necessárias, principalmente, para determinar até que ponto a nova implementação realizada representa um avanço em relação à versão anterior da UseSkill. As seções seguintes descrevem em detalhes as aplicações utilizadas, como foram organizadas as avaliações e como a ferramenta foi operada durante elas, bem como as descobertas resultantes da análise dos resultados obtidos.

4.1 Estudo Experimental

Os métodos empíricos na engenharia de software ganharam popularidade amplamente devido ao fato de os resultados obtidos virem do mundo real. Os resultados obtidos de maneira empírica com a presença de participantes são baseados no fator humano e dão uma estimativa do efeito do objeto estudado no experimento na realidade a que ele está sujeito (KUZNIARZ; STARON; WOHLIN, 2003).

Assim, tendo em vista que desejava-se estudar o efeito da EUSADUS na prática, uma avaliação experimental foi planejada e conduzida. Essa avaliação segue as diretrizes propostas por Wohlin et al. (2012) para experimentação em Engenharia de Software, e ajudou a identificar os pontos fortes e fracos da abordagem proposta.

4.1.1 Objetivo

O propósito deste estudo experimental é analisar a detecção de problemas de usabilidade com e sem a utilização da abordagem proposta neste trabalho, com o intuito de avaliar o impacto na detecção de problemas, com respeito a efetividade de descoberta de problemas, do ponto de vista do pesquisador, no contexto de desenvolvedores de software trabalhando com sistemas de informação na Web.

4.1.2 Questões de Pesquisa

O estudo foi planejado de maneira a responder às seguintes questões:

- **QP1**: a EUSADUS permite a detecção correta de *usability smells*?
- **QP2**: os *smells* detectados podem indicar problemas de usabilidade concretos?

- **QP3**: os problemas de usabilidade identificados com o auxílio da EUSADUS são os mesmos encontrados com a ajuda da UseSkill?

4.1.3 Hipóteses

De acordo com as questões de pesquisa, foram formuladas as seguintes hipóteses:

- **Hipótese nula**, $H1_0$: os problemas de usabilidade identificados pela UseSkill (P_U) e pela EUSADUS (P_E) são o mesmos, ou $H1_0$: $P_U = P_E$. **Hipótese alternativa**, $H1_1$: $P_U \neq P_E$;
- **Hipótese nula**, $H2_0$: a quantidade de problemas de usabilidade distintos identificados com auxílio da UseSkill (P_U) e com ajuda da EUSADUS (P_E) é a mesma, ou $H2_0$: $count(P_U) = count(P_E)$. **Hipótese alternativa**, $H2_1$: $count(P_U) \neq count(P_E)$;

4.1.4 Variáveis

Neste estudo, os métodos de teste de usabilidade (UseSkill e EUSADUS) representam as variáveis independentes (WOHLIN et al., 2012), ou seja, aquelas que se alternam durante a avaliação. Os problemas de usabilidade detectados, por sua vez, representam as variáveis dependentes, ou seja, aquelas que são determinadas de acordo com o efeito causado pelas variáveis independentes.

4.1.5 Objeto

O objeto utilizado foi foi um sistema de gestão de planos de saúde disponível na Web e utilizado por centenas de usuários todos os dias. Essa escolha ocorreu de acordo com a conveniência, uma vez que o grupo de pesquisa do qual o autor desta dissertação faz parte possui acesso à empresa e aos desenvolvedores de tal sistema. Ele foi analisado usando as abordagens da UseSkill e da EUSADUS.

4.1.6 Participantes

O único participante direto deste estudo foi um desenvolvedor e conhecedor da aplicação sob teste e da ferramenta UseSkill, que foi escolhido para o papel de avaliador. Essa seleção foi baseada na conveniência, pois apenas um desenvolvedor que tenha trabalhado com a versão do sistema usada encontrava-se disponível para participar da avaliação. Assim, este estudo pode ser classificada como *Quasi-experiment* (WOHLIN et al., 2012), uma vez que a atribuição dos tratamentos aos participantes não foi aleatória.

Apesar de conhecer a ferramenta UseSkill, o participante em questão a utilizou apenas em sua versão anterior, conforme apresentada em Souza et al. (2016a). Assim, ele

tem experiência sobre como criar testes, definir tarefas e analisar sessões de uso, mas não era familiar com os novos conceitos introduzidos pela abordagem proposta, como *usability smells*, *refactorings* e as estratégias de detecção utilizadas.

Indiretamente, os usuários do sistema também participaram da avaliação, uma vez que os *logs* analisados foram gerados por sua interação com a aplicação objeto. Essa interação, contudo, ocorreu durante suas atividades normais do dia a dia, sem nenhuma necessidade de movimentação ou alocação de tempo dos participantes.

4.1.7 Contexto

O contexto usado para a avaliação foi o ambiente de trabalho do próprio desenvolvedor da aplicação, nos intervalos em que ele estava disponível. Tanto a execução da busca de *smells* quanto a análise dos relatórios da ferramenta foram realizadas com uma instância local da mesma, executada dentro de seu ambiente de desenvolvimento. No caso dos usuários da aplicação, o contexto da captura de dados foi o ambiente de trabalho dos funcionários do plano de saúde e o ambiente rotineiro dos beneficiários do mesmo.

4.1.8 Desenho Experimental

O desenho experimental utilizado foi o de um fator (método de teste de usabilidade) com dois tratamentos (UseSkill e EUSADUS). Esse desenho permite não apenas comparar o desempenho da nova abordagem com outro método de avaliação de usabilidade comprovadamente eficaz, mas também verificar se a extensão realizada representa um avanço em relação a versão antiga da ferramenta.

4.1.9 Operação

O conjunto de dados usado na avaliação consiste nas ações capturadas no sistema de 7 a 21 de março de 2016, ou seja, em um período de duas semanas. Como o ciclo de vida do sistema é iterativo e incremental, essa janela de tempo corresponde a uma versão específica do mesmo, não representando o seu estado atual. Nesse período foram capturadas cerca de 400 mil ações.

Para os *smells* que utilizam como entrada as sessões de uso, é necessária a seleção das funcionalidades (tarefas) que serão analisadas. Essa seleção foi realizada com base nas funcionalidades mais utilizadas da aplicação, apresentadas na Tabela 3. A soma das ações contidas nessas quatro tarefas corresponde a cerca de 50% do total de ações capturadas. Os outros 50% correspondem às ações das funcionalidades dos sistema que não foram analisadas.

Para cada uma das funcionalidades na Tabela 3, o avaliador determinou o conjunto de ações iniciais e finais correspondentes. Em seguida, ele executou essas ações no sistema,

Tabela 3 – Funcionalidades analisadas no estudo experimental.

Id	Nome	Ações capturadas	Sessões capturadas
1	Marcar Exame	101.012	267
2	Confirmar Exame	40.739	192
3	Marcar Consulta	34.192	275
4	Solicitar Procedimento Odontológico	26.870	140

utilizando o *plugin* de auxílio para determinar automaticamente o tipo, XPath e URL das mesmas. De posse dessas informações, o cadastro do teste foi realizado na ferramenta UseSkill.

Como as duas abordagens (UseSkill e EUSADUS) utilizaram o mesmo conjunto de dados de entrada, o processo de captura de dados e definição de tarefas foi executado somente uma vez. Entretanto, a detecção de problemas de usabilidade com a UseSkill sem auxílio da abordagem proposta foi realizada buscando-se manualmente as sessões/ações problemáticas nos relatórios gerados, conforme descrito em Souza et al. (2016a), enquanto a detecção com a abordagem proposta segue os passos descritos a seguir. A avaliação realizada por Souza et al. (2016a) não será detalhada nesta seção por já ter sido apresentada em outro trabalho, contudo, as diferenças entre os resultados de ambas as abordagens serão discutidas na subseção 4.1.11.

Antes de iniciar a avaliação com a nova abordagem, o avaliador recebeu uma breve explicação sobre o conceito e o propósito de cada *usability smell* implementado. Os parâmetros utilizados por cada estratégia de detecção foram todos definidos com os valores padrão, evitando um esforço de configuração. Assim, podemos avaliar também a escolha dos limiares, de acordo com o seu desempenho em conjunto com a abordagem.

Em seguida, cada uma das tarefas na Tabela 3 foi analisada automaticamente, tentando encontrar os *smells Laborious Task*, *Cyclic Task* e *Too Many Layers*. Os *smells Lonely Action*, *Missing Feedback* e *Fat Interface*, por não exigirem a seleção de tarefas, utilizaram todo o conjunto de dados capturados em sua análise.

O *smell Undescriptive Element* não pôde ser incluído na análise devido à configuração utilizada na captura dos eventos, mais especificamente do evento *mouseover*. Esse evento exige a definição de um valor, em segundos, que determina após quanto tempo ele será disparado. Como o conjunto de dados foi previamente capturado e os valores definidos pela UseSkill, na ocasião, e pela abordagem de Grigera et al. (2017) para esse parâmetro são diferentes, não foi possível inferir a mesma conclusão de acordo com a quantidade de eventos capturados.

Os relatórios gerados ao final da etapa de análise foram, então, examinados pelo avaliador, que tentou interpretar os mesmos e determinar o que representava, ou não, problemas de usabilidade. As conclusões do avaliador sobre essa investigação são apresentadas

na subseção 4.1.10.

4.1.10 Resultados

4.1.10.1 *Laborious Task*

Os limiares calculados pela ferramenta para esse *smell* foram 54 ações e 15 minutos. Com isso, a detecção indicou problemas nas Tarefas 1 (2,62% afetada) e 4 (2,86% afetada). Para a Tarefa 1, 7 sessões foram indicadas como problemáticas, enquanto que, para a Tarefa 4 foram indicadas 4 sessões.

No caso da Tarefa 1, todas as sessões indicavam um mesmo problema, que diz respeito ao conjunto de ações que o usuário deve executar para marcar um exame. Se ele deseja marcar vários exames, esse conjunto de ações deve ser repetido um número equivalente de vezes, assim, quando há a marcação de vários exames ao mesmo tempo, um grande número de ações é necessário. Esse caso não necessariamente representa um problema, mas essa atividade poderia ser otimizada, definindo-se o valor padrão das caixas de seleção com o valor mais utilizado, por exemplo.

De forma análoga, todas as sessões da Tarefa 4 também indicavam um mesmo problema. Esse problema, já identificado anteriormente na avaliação da ferramenta UseSkill (SOUZA et al., 2016a), ocorre ao solicitar um mesmo procedimento odontológico para vários dentes. O usuário é obrigado a preencher um grande número de informações para cada um deles, em vez de poder apenas preenchê-las e escolher os dentes aos quais deseja aplicar essas mesmas informações. Apesar de semelhante ao caso da Tarefa 1, esse processo ocasiona grande esforço devido à má organização das ações que devem ser executadas (ou seja, por um erro dos desenvolvedores da aplicação), não pela natureza do procedimento.

Pode-se notar, em geral, que houve muitas detecções redundantes. Contudo, como a quantidade de sessões selecionadas para análise é pequena, esse fator não representa um grande problema. Além disso, parte das sessões foi indicada como similar pela própria ferramenta, utilizando-se a funcionalidade de agrupamento de sessões similares.

4.1.10.2 *Cyclic Task*

Para esse *smell*, a ferramenta detectou um total de 183 sessões, sendo 104 referentes à Tarefa 1 (38,95% afetada), 7 à Tarefa 2 (3,65% afetada), 7 à Tarefa 3 (2,55% afetada) e 65 à Tarefa 4 (46,43% afetada). Devido ao grande número de sessões, tornou-se inviável realizar a análise de todas manualmente. Assim, um conjunto de 5 sessões aleatórias foi escolhido para cada tarefa e analisado pelo avaliador. Em todos os casos, contudo, as sessões referentes a uma mesma tarefa indicaram um mesmo problema, resultando, assim, na identificação de 4 problemas diferentes.

Na Tarefa 1, observou-se que os usuários sempre preenchiam o componente de selecionar exame mais vezes que a quantidade. Cada exame precisa de uma quantidade definida, assim, a diferença entre a quantidade de vezes em que essas duas ações foram executadas corresponde à quantidade de vezes em que o usuário errou ao tentar buscar um exame. Esse é um claro sinal de confusão do usuário, assim, dar sugestões com base nos exames mais solicitados pelo médico, por exemplo, poderia ajudar a reduzir a quantidade de ações na tarefa.

Na Tarefa 2, observou-se a execução de várias ações similares em sequência. A sessão da Figura 20, que exemplifica esse problema, apresenta um ciclo (à direita) onde o usuário selecionou mais de 10 elementos para autorizar exames solicitados (o tamanho de cada nó é proporcional à quantidade de vezes em que a ação que ele representa foi executada), indicando que não há um botão "selecione todos" na interface. Esse problema pode passar facilmente despercebido quando a quantidade de elementos que precisam ser selecionados é pequena, mas é bem evidente em casos como esse.

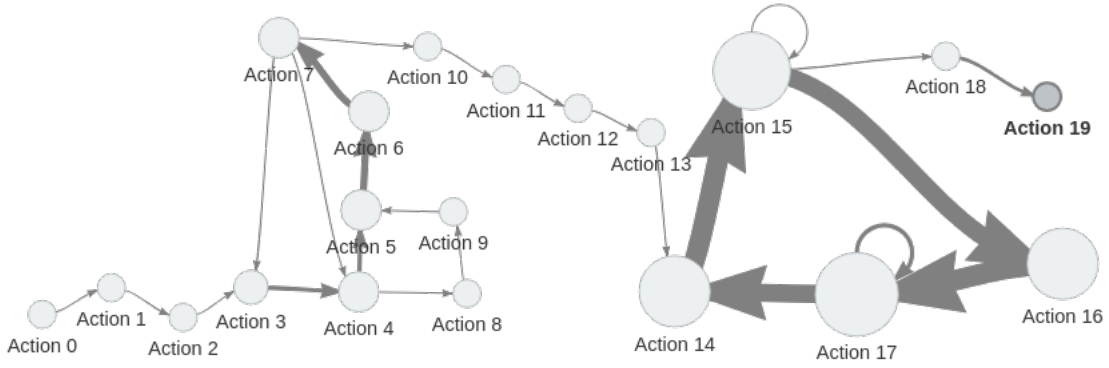


Figura 20 – Sessão indicando um problema na Tarefa 2. Fonte: Autor (2018).

Na Tarefa 3, observou-se que uma determinada ação foi executada várias vezes seguidas, como pode ser visto na sessão da Figura 21. Ao selecionar um médico, o campo de especialidades é carregado, entretanto, não há indicação para o usuário sobre quando esse carregamento está ocorrendo.

Assim, o usuário clicou várias vezes no campo de seleção de médico (*Action* 10), até a lista de especialidades ser carregada. Esse problema pode ser facilmente solucionado, bastando indicar ao usuário que um carregamento de dados está ocorrendo, com o uso de uma barra de progresso, por exemplo. Apesar de todas as sessões analisadas para essa tarefa terem indicado a mesma falta de *feedback*, esse problema aparece em diferentes elementos do sistema entre elas.

Na Tarefa 4, observou-se um ciclo de ações que repetia-se um grande número de vezes, com apenas uma ação diferente entre cada um desses ciclos de repetição. Esse problema também foi identificado durante a análise das sessões detectadas para o *smell Laborious Task*, e ocorre pois o usuário precisa selecionar o tratamento, face, arcada,

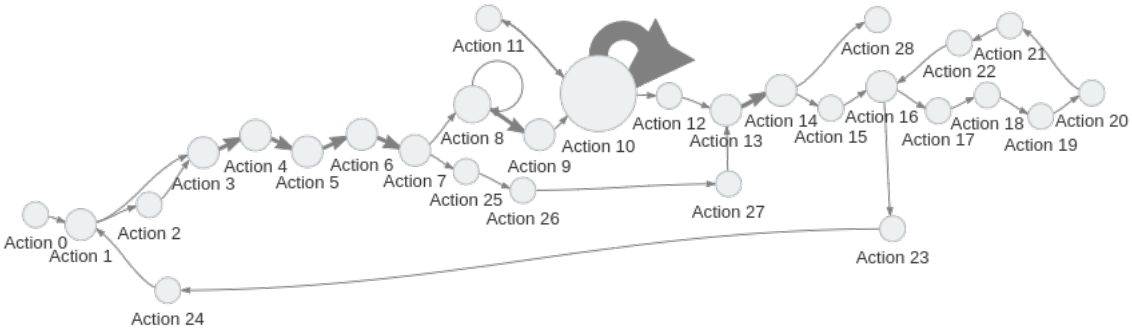


Figura 21 – Sessão indicando um problema na Tarefa 3. Fonte: Autor (2018).

quantidade, etc. para cada elemento dentário a ser tratado. Ou seja, se ele deseja solicitar um mesmo tratamento para 9 dentes diferentes, esse procedimento terá de ser repetido para cada um deles. Como também já foi discutido, esse problema pode ser solucionado com a reorganização do fluxo de ações da tarefa.

A grande quantidade de sessões detectadas deve-se principalmente à proporção em que as tarefas foram afetadas. No caso da Tarefa 4, por exemplo, quase metade dos usuários enfrentou o problema em questão, o que sugere prioridade na resolução desse problema. Apesar de nem todas as sessões problemáticas terem sido analisadas, levando-se em conta o comportamento observado é possível inferir que grande parte delas tenha sido afetada pelos mesmos problemas já apontados.

4.1.10.3 *Lonely Action*

A ferramenta indicou 5 ações contendo esse *smell*. A primeira delas corresponde ao clique em um link que foi executado 298 vezes, representando um total de aproximadamente 71% das ações realizadas na página. Esse link redireciona o usuário para a página de solicitação de um novo exame. Toda vez que o usuário solicita um exame corretamente, é apresentada a ele uma página de sucesso, onde o link de redirecionamento aparece.

O profissional (usuário) que trabalha com a solicitação de exames tende a executar esse procedimento várias vezes em sequência, pois atende vários clientes com o mesmo objetivo. Assim, toda vez que ele finaliza uma solicitação, é obrigado a executar a ação detectada para poder solicitar um novo exame. Ao concluir uma solicitação, o usuário deveria ser automaticamente redirecionado à página de nova solicitação, onde a mensagem de sucesso poderia ser exibida.

A segunda ação foi executada 603 vezes, representando cerca de 92% das ações realizadas na página. O problema associado à essa ação é bem semelhante ao encontrado na primeira. Após gerar um cartão de beneficiário, o usuário é redirecionado a uma tela de sucesso e precisa clicar em outro elemento para cadastrar outro cartão. Nesse caso, ele também deveria ser redirecionado automaticamente para a página contendo a mensagem

de *feedback* e o formulário para cadastro de um novo cartão.

Para as demais ações detectadas, o avaliador afirmou que não conhecia as partes dos sistema envolvidas, assim, os problemas não puderam ser identificados. Apesar de tudo, os resultados apresentados ajudaram a identificar atividades que poderiam ser facilmente automatizadas, sem a necessidade de analisar nenhuma sessão de uso. Além disso, não houveram problemas com redundância, e a análise pôde ser realizada sobre todo o conjunto de dados.

4.1.10.4 *Too Many Layers*

Para esse *smell* foram detectadas 2 sessões para a Tarefa 1 (0,75% afetada), 1 sessão para a Tarefa 2 (0,52% afetada), 8 sessões para a Tarefa 3 (2,91% afetada) e 1 sessão para a Tarefa 4 (0,35% afetada). Todas as sessões analisadas indicaram problemas já identificados anteriormente através da análise das detecções indicadas para os *smells* *Laborious Task* e *Cyclic Task* ou problemas cuja origem não pôde ser identificada.

O avaliador também informou que a forma de visualização dos resultados para esse *smell* era ruim, e que seria melhor ter somente uma lista com as URLs por onde o usuário passou, em vez das sessões de uso. Assim, seria mais fácil visualizar o caminho percorrido pelo usuário, que é o foco do problema apontado por esse *smell*, sem ter que percorrer toda a sequência de ações executadas por ele.

4.1.10.5 *Missing Feedback*

Foram indicados 2 elementos do sistema contendo esse *smell*. O primeiro deles corresponde à célula de uma tabela, com uma mediana de 6 cliques e um total de 10 instâncias, para o qual o avaliador não conseguiu apontar a causa do problema, ou seja, ele não entendeu o que levou os usuários a adotarem o comportamento identificado. Assim, esse caso não foi considerado um problema.

O segundo corresponde a um elemento do odontograma, com mediana de 4 cliques e um total de 6 instâncias. Para esse caso, o avaliador afirmou que, aparentemente, muitos usuários tentaram selecionar um determinado dente para marcar sua situação, mas o componente com as opções disponíveis não estava aparecendo. De acordo com ele, isso pode ter sido causado por algum *bug* no código da parte gráfica da página, como CSS ou *JavaScript*.

4.1.10.6 *Fat Interface*

Inicialmente, esse *smell* foi detectado em 16 páginas diferentes da aplicação. Contudo, após uma análise das mesmas, constatou-se que se tratavam de páginas que utilizavam um componente que realiza alterações dinâmicas na interface. Desse modo, cada uma das

URLs apontadas pode estar associada à várias interfaces diferentes e, consequentemente, todos os elementos interagidos de cada uma dessas interfaces são contados, o que vai contra a definição desse *smell*, gerando uma contagem incorreta de ações. Assim, nenhum problema pôde ser identificado com esse *smell*.

4.1.11 Discussões

Considerando os resultados obtidos, a resposta para a QP1 é afirmativa: a EUSADUS permitiu a detecção correta de *usability smells*. Com o *Laborious Task* foram identificadas duas tarefas que exigiam grande esforço; com o *Cyclic Task* foram identificadas duas tarefas altamente repetitivas; com o *Lonely Action* foram identificadas duas ações que eram quase sempre as únicas executadas nas interfaces em que estavam presentes; com o *Too Many Layers* foram indicadas algumas formas de execução de todas as tarefas que levaram os usuários a percorrer um longo caminho; e com o *Missing Feedback* foi apontado um elemento clicado várias vezes porque os usuários não recebiam resposta do sistema.

Para a QP2, por sua vez, a resposta também é afirmativa: os *smells* indicados ajudaram a encontrar problemas de usabilidade concretos em uma aplicação real. Os *smells Laborious Task*, *Cyclic Task* e *Too Many Layers* ajudaram a identificar problemas de organização, falta de *feedback* e flexibilidade, e confusão do usuário (NIELSEN, 1993), no fluxo das tarefas; o *Lonely Action* ajudou a identificar ações redundantes; e o *Missing Feedback* ajudou a identificar um *bug* na interface gráfica.

Por fim, a resposta para a QP3 é negativa: problemas diferentes foram detectados em ambas as abordagens. Nota-se que alguns dos problemas identificados anteriormente pela UseSkill não puderam ser encontrados através da análise dos relatórios dos *usability smells* detectados. Esse comportamento já era esperado, uma vez que a abordagem proposta abre mão de algumas etapas extras de configuração utilizadas pela UseSkill que facilitam a detecção de problemas. Entretanto, também é importante apontar que diversos novos problemas foram identificados através da análise dos *smells*. Assim, a hipótese $H1_0$ foi rejeitada.

A quantidade de problemas identificados também foi diferente entre as avaliações. A avaliação da UseSkill identificou 10 problemas de usabilidade distintos (SOUZA et al., 2016a), enquanto que, com a EUSADUS foram identificados 7 problemas distintos. Vários fatores influenciaram nessa diferença, como o fato de nem todos os *smells* definidos terem sido incluídos na detecção e a falta de conhecimento do avaliador sobre algumas partes do sistema, por exemplo. Assim, a hipótese $H2_0$ foi rejeitada.

Uma das principais metas que guiaram o desenvolvimento da EUSADUS foi a de que ela deveria, no mínimo, manter um poder de auxílio à detecção de problemas de usabilidade semelhante ao da UseSkill, enquanto simplificando a sua operação. A

simplicidade de utilização de ambas não foi avaliada, devido às limitações referentes ao avaliador.

Contudo, a EUSADUS, além de requerer menos configurações que a UseSkill, aponta o que deve ser analisado automaticamente e ainda auxilia teoricamente a análise. Além disso, os resultados obtidos mostram que alguns dos problemas identificados com auxílio da EUSADUS afetam uma grande quantidade de usuários e geram muito desperdício de tempo e esforço dos mesmos. Em adição, a implementação de novos *smells* pode aumentar ainda mais seu poder de auxílio. Contudo, mais avaliações são necessárias para confirmar se há, de fato, uma diferença significativa entre as duas abordagens.

4.1.12 Ameaças à Validade

4.1.12.1 Validade de Conclusão

Foram identificados dois fatores, além dos mensurados nesta avaliação, que podem ter influenciado nas conclusões obtidas. O primeiro deles é referente as métricas utilizadas. Somente foram comparados a quantidade de problemas e a unicidade dos mesmos. A quantidade de problemas por si não é uma medida representativa para se mensurar a significância de uma abordagem de detecção de problemas de usabilidade, uma vez que esses problemas podem ter diferentes níveis de impacto.

O impacto dos problemas identificados não foi mensurado pois isso requer o auxílio de ao menos dois especialistas em usabilidade, para garantir que não há viés nessa definição. Infelizmente, o autor desta dissertação não conseguiu acesso a tais profissionais no período da pesquisa. Contudo, através das métricas geradas pela ferramenta para cada um dos *smells* (como a proporção em que a tarefa foi afetada, por exemplo) é possível ter uma noção da quantidade de usuários afetados pelo problema. Como pôde ser visto, um dos problemas identificados afetou quase 50% dos usuários, enquanto outro evidenciou uma ação desnecessária que foi executada mais de 600 vezes, dando assim uma indicação de que a EUSADUS pode ajudar a identificar problemas de grande impacto.

Segundo, os dados capturados são referentes à uma versão do sistema de mais de 2 anos atrás. Enquanto a avaliação realizada por Souza et al. (2016a) ocorreu no mesmo ano em que os dados foram capturados, a avaliação com a abordagem proposta ocorreu somente no ano corrente, e não foi possível ter acesso à versão do sistema em questão. Assim, pode-se afirmar que o nível de experiência do desenvolvedor/avaliador com o sistema analisado era diferente nas duas ocasiões. Contudo, o fato de que muitos problemas foram identificados sem a necessidade de acesso direto ao sistema demonstra que o avaliador ainda tem experiência com o mesmo. Além disso, a abordagem proposta foi o tratamento colocado em "desvantagem", nesse caso.

4.1.12.2 Validade Interna

Em relação aos fatores extras que podem ter influenciado nos resultados, destaca-se o contexto da avaliação. Devido à disponibilidade limitada do desenvolvedor/avaliador, a avaliação teve que ser dividida em várias partes, de acordo com quando o mesmo estava disponível, durante alguns horários livres em seu trabalho. Assim, a perda de foco pode ter influenciado no seu desempenho.

Para tentar minimizar a influência das interrupções, o avaliador focou-se nos relatórios sobre as partes do sistema que ele tinha mais familiaridade. Isso permitiu que ele identificasse origem dos *smells* detectados com mais facilidade, aumentando assim as chances de detectar mais problemas no tempo limitado que ele tinha disponível.

Também devido às interrupções, a duração total da avaliação não foi medida, já que assim essa métrica não representaria realisticamente uma avaliação contínua. Além disso, o tempo demandado para realizar a análise de resultados pode variar muito, dependendo do conhecimento do avaliador sobre o sistema e de sua experiência com usabilidade, e esses fatores não foram mensurados nesta avaliação devido à disponibilidade de um único avaliador. Esses fatores devem ser considerados em avaliações futuras para embasar melhor a eficácia da abordagem proposta.

Outro fator que pode ter influenciado nos resultados é fato de o avaliador ter conhecimentos sobre o funcionamento da ferramenta UseSkill (ou seja, sobre a definição de tarefas, sessões e como devem ser analisadas). Isso pode ter melhorado seu desempenho na análise, o que não ocorreria com um desenvolvedor que conhecesse apenas o sistema analisado. Contudo, os conceitos utilizados pela ferramenta são bem simples e podem ser facilmente aprendidos através de uma curta apresentação.

4.1.12.3 Validade de Constructo

Sobre a influência que pode ter sido causada pela forma de construção da avaliação, destaca-se a escolha do avaliador. Um mesmo avaliador aplicou os dois tratamentos utilizados (UseSkill e EUSADUS). Assim, o segundo tratamento aplicado pode ter sido beneficiado pela experiência adquirida com a aplicação do primeiro. Para analisar esse efeito, foi realizada a comparação entre os problemas encontrados por ambas as avaliações.

Apenas um dos problemas identificados foi comum entre as duas avaliações. A detecção de *smells* considera aspectos diferentes dos analisados por Souza et al. (2016a), o que significa que o avaliador foi guiado a explorar, em maior parte, comportamentos não analisados na avaliação anterior. Em adição, de acordo com o avaliador, o problema comum com a solicitação de procedimentos odontológicos já era bem conhecido do time de desenvolvimento, mesmo antes da primeira avaliação realizada.

4.1.12.4 Validade Externa

Entre os fatores que podem dificultar a capacidade de generalização dos resultados estão a quantidade de participantes e o conjunto de dados analisados. Apenas um desenvolvedor pôde participar da avaliação, o que significa que não foi possível determinar o efeito do nível de experiência do mesmo.

Em relação à experiência com usabilidade, a própria ferramenta já busca minimizar esse fator, provendo explicações sobre o conceito de cada *smell* (Figura 19a) e *refactorings* (Figura 14) para ajudar os desenvolvedores menos experientes a identificá-los e corrigi-los. Em relação à experiência com o sistema, como pôde ser visto pelos resultados, o desenvolvedor participante, apesar de experiente, não conhecia o sistema completo. Além disso, o ideal é que a avaliação seja realizada com múltiplos desenvolvedores diferentes, para que se tenha uma maior cobertura na identificação de problemas.

O conjunto de dados utilizado para a avaliação foi o mesmo que já havia sido analisado com auxílio da UseSkill, e cujos problemas inspiraram a criação de dois dos *smells* analisados (*Laborious Task* e *Cyclic Task*). Assim, isso pode ter direcionado as estratégias de detecção para funcionarem nesse sistema específico. Apesar disso, novos problemas foram detectados a partir dos *smells* definidos, indicando que eles não estão limitados à ideia através da qual foram definidos. Além disso, como será visto pelos resultados da segunda avaliação, a ferramenta também foi capaz de detectar *usability smells* que indicaram problemas concretos em um sistema diferente.

4.2 Estudo de Viabilidade

Qualquer processo de engenharia requer *feedback* e avaliação. O desenvolvimento de software é uma disciplina da engenharia e a mensuração das capacidades é o mecanismo ideal para se obter *feedback* sobre e avaliar o resultado desse processo. As informações obtidas através da medição ajudam a tomar decisões inteligentes e melhorar o produto desenvolvido com o tempo (BASILI, 1992).

A primeira avaliação realizada ajudou a mensurar a eficácia e eficiência da EUSADUS em comparação ao método de teste de usabilidade da UseSkill. Contudo, essa avaliação foi realizada em um cenário próximo ao ideal, onde a quantidade de dados analisados teve pouca interferência na operação da ferramenta, e o tamanho do sistema era suficiente para que um único avaliador o conhecesse quase por completo. Considerando que a proposta deste trabalho é que sua abordagem possa ser utilizada com todos os tipos de aplicação Web, é necessário obter *feedback* sobre os mais diversos cenários em que ela possa ser aplicada, especialmente aqueles que envolvem situações desfavoráveis à mesma.

Assim, uma segunda avaliação foi realizada para medir o desempenho da ferramenta

em um cenário de maior proporção. Essa avaliação visa determinar os limites, em termos de poder computacional, da extensão implementada, levando em conta os principais aspectos que podem inviabilizar sua utilização.

4.2.1 Objetivo

O objetivo deste estudo, descrito de acordo com o paradigma GQM (*Goal, Question, Metric*) (BASILI, 1992) é analisar a extensão da ferramenta UseSkill com o propósito de avaliar em relação ao desempenho computacional do ponto de vista do pesquisador no contexto de usuários interagindo com uma aplicação Web.

4.2.2 Questões de Pesquisa

Para guiar a avaliação no sentido de atingir o objetivo do estudo, as seguintes questões foram elaboradas:

- **QP1**: a quantidade de ações que podem ser analisadas pela EUSADUS é limitada?
- **QP2**: a quantidade de sessões que podem ser analisadas pela EUSADUS é limitada?
- **QP3**: a EUSADUS pode ser utilizada para analisar uma aplicação Web de grande porte?

4.2.3 Objeto

O Sistema Integrado de Gestão de Atividades Acadêmicas da Universidade Federal do Piauí (SIGAA/UFPI) informatiza os procedimentos da área acadêmica da Universidade Federal do Piauí. Esse sistema conta com diversos módulos, abrangendo desde o ensino infantil até a pós graduação, e possui milhares de usuários e milhões de linhas de código, permitindo o controle de diversas atividades como projetos de pesquisa, ações de extensão, projetos de ensino, registro e relatórios da produção acadêmica, ensino a distância, etc. (FILHO; AQUINO; ROSA, 2013)

4.2.4 Participantes

Os participantes diretos desta avaliação foram um desenvolvedor do sistema e o autor desta dissertação. Conforme será melhor detalhado na subseção 4.2.6, esta avaliação foi dividida em duas partes. A primeira delas foi conduzida pelo autor, focando-se na descoberta dos limites da EUSADUS. A segunda foi executada com auxílio do desenvolvedor, focando-se na análise do desempenho da ferramenta com um sistema de grande porte.

A escolha do autor foi baseada no nível de experiência, uma vez que era necessário conhecimento sobre a implementação da ferramenta para realizar alguns dos testes

conduzidos na primeira parte. A escolha do desenvolvedor, entretanto, foi baseada na conveniência, pois apenas um dos membros do time de desenvolvimento foi disponibilizado pela gerência do sistema.

Os usuários do sistema também participaram, mas indiretamente, desta avaliação, já que os *logs* analisados foram gerados por suas ações de interação. Essa participação pode ser considerada como aleatória, uma vez que a captura foi realizada com todos os usuários no período especificado, sem nenhum tipo de controle sobre o perfil dos mesmos.

4.2.5 Contexto

O contexto desta avaliação foi, para o caso do autor, seu ambiente de pesquisa do dia a dia e, para o caso do desenvolvedor, seu ambiente de trabalho. Esse é o único local onde é possível ter acesso ao ambiente de teste do sistema, o que foi necessário para a execução deste estudo, sendo assim o ambiente mais adequado para o mesmo.

O contexto dos usuários da aplicação foi o seu ambiente de trabalho, no caso dos funcionários da instituição que utiliza o sistema objeto, e o ambiente do dia a dia, no caso dos estudantes da instituição de ensino. Todas as operações realizadas com a EUSADUS foram executadas em uma instância local da mesma, rodando em um computador pessoal com processador Quad Core Intel® i7-3630QM de 2.40GHz e memória RAM de 4 GB dedicada para a operação da ferramenta.

4.2.6 Operação

O conjunto de dados usado na avaliação consiste nas ações capturadas no sistema no período de 28 de janeiro à 28 de fevereiro de 2018, ou seja, em um intervalo de 32 dias. Esse intervalo foi planejado para englobar o período de matrículas na instituição, que representa um dos maiores picos no uso do sistema durante todo o ano. Além disso, ele também cobre os 3 primeiros dias de aula das turmas de graduação, o que representa outro período de uso intenso do sistema, ainda que em menor proporção. Isso tudo resultou em um total de cerca de 11 milhões de ações capturadas.

As funcionalidades para a análise foram escolhidas para representar algumas das atividades mais executadas nesse período, definidas de acordo com a experiência do desenvolvedor e com uma funcionalidade previamente implementada na UseSkill que indica as URLs mais acessadas nos *logs* capturados. A Tabela 4 apresenta essas funcionalidades, juntamente com a quantidade de ações e sessões analisadas para cada uma delas[1].

[1] A quantidade de ações e sessões capturadas para a funcionalidade “Realizar Matrícula” corresponde a um intervalo de 12 horas, enquanto que, para as demais, todos os 32 dias são contados. Essa restrição foi necessária devido a limitação dos recursos de hardware utilizados nesta avaliação. Mais detalhes na subseção 4.2.7.1.

Tabela 4 – Funcionalidades escolhidas para o estudo de viabilidade.

Id	Nome	Ações analisadas	Sessões analisadas
1	Realizar Matrícula	104187	5953
2	Inscrever-se em Monitoria	248414	10715
3	Cadastrar Notas	19513	536
4	Lançar Frequência	13322	506

A determinação das ações iniciais e finais, contudo, não pôde ser realizada com auxílio do *plugin* da UseSkill. O SIGAA/UFPI é um sistema sazonal, o que significa que algumas funcionalidades do mesmo só estão disponíveis durante um certo período do ano, mesmo em ambiente de teste. Como a avaliação foi realizada em um período diferente da captura dos dados, as páginas correspondentes ao início de algumas funcionalidades não puderam ser acessadas e, consequentemente, as ações iniciais presentes nelas não puderam ser executadas para que o *plugin* determinasse suas identificações. Desse modo, as ações tiveram que ser identificadas através de uma busca manual no banco de dados de ações capturadas. Procurando pelo texto presente nos elementos que iniciavam e finalizavam cada funcionalidade escolhida, foi possível localizar os dados de identificação dessas ações.

Após a definição de tarefas, esta avaliação foi dividida em duas etapas. A primeira delas, conduzida pelo autor desta dissertação, consistiu em realizar diversos testes com a ferramenta, utilizando diferentes configurações, para analisar seu comportamento, limites e possibilitar a execução da segunda etapa (mais detalhes na subseção 4.2.7). A segunda, conduzida com um dos desenvolvedores da aplicação, teve por objetivo responder a QP3. Ela consistiu em executar a detecção de *smells*, dentro das limitações identificadas na primeira etapa, e verificar se o avaliador conseguia identificar problemas de usabilidade a partir dos relatórios gerados.

De forma análoga ao que ocorreu na primeira avaliação descrita nesta dissertação, o desenvolvedor/avaliador recebeu uma breve explicação sobre os conceitos utilizados pela ferramenta antes de iniciar a avaliação. A definição de parâmetros, por sua vez, ocorreu de acordo com os resultados da primeira etapa, que serão discutidos em detalhes na subseção 4.2.7. Nesta avaliação, a captura do evento *mouseover* ocorreu de acordo com o definido por Grigera et al. (2017), assim todos os *smells* catalogados fizeram parte da detecção.

4.2.7 Resultados

4.2.7.1 Limitações Identificadas

Inicialmente, tentou-se realizar a detecção de *smells* com todos as ações capturadas nos 32 dias e todas as tarefas definidas. Em seguida, tentou-se utilizar diferentes combinações de tarefas, *smells* e janelas de tempo para determinar qual seria mais viável. Todos

esses testes realizados resultaram em erros por falta de memória ou foram cancelados manualmente, devido ao alto tempo de processamento, que em alguns casos chegou a mais de duas horas sem finalizar a execução.

Assim, concluiu-se que, para que a detecção pudesse ser executada, seria necessário restringi-la bastante. Estabeleceu-se, então, algumas restrições baseadas no conhecimento do autor sobre o funcionamento da ferramenta e em mais testes realizados posteriormente para estabelecer quais seriam as configurações ideias. As restrições estabelecidas foram:

1. **O Banco de Dados deve ser indexado**. As ações armazenadas no BD possuem uma série de atributos associados a elas que ajudam em sua caraterização. Contudo, eles se tornam um problema quando é necessário buscar uma informação específica com uma quantidade tão grande de registros como a que foi capturada para essa avaliação. A indexação é um processo que ajuda a acelerar muito as consultas que envolvem atributos com alta cardinalidade. Assim, os atributos de maior cardinalidade das ações presentes no BD, como marca temporal (*timestamp*) e ID do usuário, foram manualmente indexados para reduzir o tempo gasto com consultas ao BD;

2. **Apenas uma ação inicial e uma ação final pode ser definida para cada tarefa**. A quantidade de ações iniciais e finais influencia no tempo necessário para agrupar as sessões, assim, reduzi-las ao mínimo possível diminui o tempo de gasto na etapa de análise dos dados;

3. **As estratégias de detecção de *smells* que utilizam sessões como entrada devem ser executadas separadamente das que usam ações**. Quando a detecção é realizada com ambos os tipos de estratégia, tanto as ações como as sessões capturadas na janela de tempo são armazenadas na memória ao mesmo tempo. Assim, separar essas detecções ajuda a reduzir o uso da memória;

4. **A tarefa "Realizar Matrícula" deve ser analisada somente com as ações referentes à 12 horas de captura**. A quantidade de ações capturadas pertencentes à essa tarefa foi bem superior às demais, o que já era esperado devido ao período em que a captura foi realizada. Contudo, mesmo quando o intervalo de tempo foi definido para apenas o primeiro dia de matrículas, o processamento resultou em erro por falta de memória. Assim, a janela temporal usada para a análise dessa tarefa corresponde às 12 horas iniciais do primeiro dia do período de matrículas no sistema (29/01). A quantidade de ações e sessões apresentadas na Tabela 4 para essa tarefa corresponde à esse intervalo definido, enquanto que para as demais tarefas corresponde à todo o período de 32 dias. Além disso, essa tarefa foi analisada separadamente das demais tarefas para evitar mais erros por falta de memória;

5. **Só podem ser analisadas até 800 mil ações simultaneamente**. Sempre que o carregamento de aproximadamente 830 mil ações ou mais era realizado simultaneamente, o processamento resultava em falhas. Assim, o limite de 800 mil ações foi estabelecido programaticamente na ferramenta, dando uma margem de erro segura para evitar a geração de mais falhas de execução. Consequentemente, todas as estratégias de detecção que usam ações como entrada analisaram as 800 mil primeiras ações capturadas.

Através dos testes preliminares foi possível identificar que as principais limitações da extensão implementada estão relacionadas à quantidade de memória RAM e à capacidade de processamento necessárias. Assim, talvez seja possível analisar um sistema de grande porte sem o estabelecimento de restrições, contudo, a quantidade de recursos computacionais necessários para essa tarefa deve ser bem elevada.

A restrição 5 é única que trata de um problema não relacionado diretamente à grande quantidade de ações para análise. Através da investigação da parte de implementação relacionada à busca de ações no BD, verificou-se que todas as ações estavam sendo carregadas simultaneamente na memória. Devido às limitações da linguagem de programação utilizada, essa ação causa um uso excessivo do coletor de lixo, que passa a consumir mais de 90% de todo o processamento disponível para a aplicação, disparando assim, o erro de processamento em questão.

Alterar a forma de carregamento das ações poderia possibilitar a análise de uma quantidade maior de ações. Contudo, atualmente, essa alteração requer uma mudança profunda na forma de operação não só da implementação abordagem proposta, mas também da própria ferramenta UseSkill. Assim, é necessário que antes seja realizado um estudo da ferramenta para que essa alteração seja feita sem a geração de problemas.

Com todas as restrições estabelecidas, a detecção de *smells* pôde, enfim, ser executadas sem erros e completando seu processamento. Logo, o próximo passo foi medir o desempenho da EUSADUS, em termos de tempo de processamento, para avaliar se ele seria aceitável, e verificar se a abordagem poderia auxiliar a detecção de *smells* mesmo com as restrições.

4.2.7.2 Desempenho Medido e *Usability Smells* Detectados

Conforme as restrições estabelecidas, a execução da detecção de *smells* teve de ser dividida em 3 partes. A primeira foi realizada buscando-se detectar os *smellls Laborious Task*, *Cyclic Task* e *Too Many Layers* nas tarefas 2, 3 e 4, apresentadas na Tabela 4, com a janela temporal cobrindo todos os 32 dias de captura. Essa detecção durou aproximadamente 23 minutos e 30 segundos, dos quais 30 segundos foram gastos com o agrupamento das sessões da Tarefa 4, 1 minuto com a Tarefa 3 e 22 minutos com a Tarefa 2. A busca de

smells nas sessões demorou menos de 1 segundo. A segunda parte foi executada buscando-se esses mesmos *smells* na Tarefa 1, com os dados referentes às primeiras 12 horas do primeiro dia de matrícula, e durou aproximadamente 12 minutos.

A terceira, e última, parte foi executada buscando-se detectar os *smells Lonely Action*, *Undescriptive Element*, *Missing Feedback* e *Fat Interface* nas 800 mil primeiras ações capturadas. Essa parte durou aproximadamente 1 minuto. Todos os parâmetros utilizados para detecção nesta avaliação foram definidos com seus valores padrão, com exceção da "quantidade de de ações máxima" e da "duração máxima", parâmetros do *smell Laborious Task*, que foram definidos com os valores identificados na primeira avaliação (54 e 15, respectivamente).

Fazendo uma análise rápida das sessões detectadas, percebeu-se que o agrupamento foi prejudicado, em alguns casos. Devido à limitação da quantidade de ações iniciais/finais, algumas sessões que deveriam ter sido classificadas como reinício foram agrupadas como parte de outras sessões, gerando uma média de quantidade de ações alta. Assim, o cálculo do valor padrão dos parâmetros do *Laborious Task* ficou comprometido, o que obrigou à uma definição manual. Também por esse motivo, a quantidade de detecções de *smells* realizadas pela ferramenta não foi mensurada.

As restrições estabelecidas geraram ruídos nos dados, reduzindo a precisão da detecção realizada. Além disso, o objetivo desta avaliação não é medir a precisão da mesma, mas sim a eficácia da EUSADUS no auxílio à detecção de problemas quando colocada em situação de "estresse". Assim, com esse objetivo em mente, decidiu-se por guiar a análise de acordo com a experiência do avaliador com o sistema.

O avaliador foi orientado à analisar apenas um pequeno número de sessões de cada tarefa, para cada *smell* detectado. E, além disso, não mais que 10 ações detectadas para cada *smell* associado à ações. A escolha das sessões e ações analisadas foi feita focando-se nos *smells* detectados nas partes do sistema que o avaliador tinha mais experiência. Dessa forma, a chance de detectar mais problemas em menos tempo aumenta. Os resultados dessa análise são discutidos a seguir. Essa segunda etapa da avaliação, incluindo o tempo de processamento da ferramenta na busca de *smells*, durou cerca de 2 horas.

4.2.7.2.1 *Laborious Task*

Na Tarefa 1, foi observado em uma das sessões que o usuário tentou clicar 7 vezes em uma turma que possuía reserva de vagas. Provavelmente, ele não entendeu que não podia realizar a matrícula no componente e insistiu várias vezes, mostrando um claro sinal de confusão do usuário. Um comportamento similar também foi identificado em outras sessões analisadas.

Na Tarefa 2, foi observado em várias sessões que os usuários preencheram vários

campos de uma tabela mais de uma vez. Em uma sessão específica, com 218 ações realizadas e 29 minutos de duração, o usuário chegou a preencher os referidos campos 4 vezes antes de conseguir concluir a execução da tarefa. O avaliador não conseguiu identificar o que exatamente causou esse problema, mas confirmou que os usuários certamente estão enfrentando alguma dificuldade na etapa de preenchimento do formulário de inscrição em monitoria.

4.2.7.2.2 *Cyclic Task*

Nas sessões detectadas com esse *smell* não foi encontrado nenhum problema de usabilidade específico. Contudo, observou-se vários comportamentos "incomuns" que geram altas taxas de ciclos, como usuários que clicam várias vezes no botão "Salvar" após cadastrar notas (provavelmente por receio de perder o trabalho realizado devido à instabilidades no sistema) e usuários que sempre clicam no plano de fundo do sistema antes de realizar o preenchimento de um campo de texto.

4.2.7.2.3 *Lonely Action*

O avaliador não conseguiu identificar nenhum problema nas ações detectadas com esse *smell*. Isso ocorreu porque ele, para cada ação analisada, ou não pôde inferir o que ocasionou o comportamento, ou desconhecia as partes do sistema afetadas.

4.2.7.2.4 *Too Many Layers*

Todas as sessões analisadas com esse *smell* foram detectadas devido ao problema com o agrupamento de sessões causado pela limitação da quantidade de ações iniciais e finais. Em vários casos, o usuário começava a executar a tarefa, mas interrompia essa execução para realizar outras atividades no sistema e depois retornava para finalizá-la através de outra ação inicial (ou seja, outro ponto de acesso à funcionalidade no sistema). Normalmente, a primeira parte dessa sessão seria classificada como "reinício", mas como a outra ação inicial em questão não foi mapeada, elas foram agrupadas como sendo parte de uma mesma sessão. Assim, a quantidade de URLs percorridas pelo usuário em todo esse processo de entrar e sair da funcionalidade é erroneamente contada como fazendo parte de uma única tentativa de execução da tarefa.

4.2.7.2.5 *Undescriptive Element*

Dois problemas foram identificados pelo avaliador nos elementos apontados com esse *smell*. Primeiramente, na tela de realização de matrícula, os usuários parecem estar confusos com um link para retornar ao portal discente. Além disso, também durante

a realização da matrícula, os usuário não parecem entender o significado do ícone de adicionar turmas, já que constantemente tentam obter uma *tooltip* dele.

4.2.7.2.6 *Missing Feedback*

Também foram identificados dois problemas associados à esse *smell*. O primeiro deles corresponde a um *bug* do sistema, durante a realização da matrícula. O ícone de "Ver Detalhes da Turma" deveria exibir uma tela com informações sobre a turma quando clicado, entretanto, essa ação estava ocasionando um erro no sistema, e nenhuma resposta era exibida para o usuário sobre isso, o que o fazia tentar clicar várias vezes nesse elemento.

Provavelmente em consequência do primeiro problema identificado, um segundo problema foi gerado. Como não conseguiam obter informações sobre a turma através do ícone correspondente, os usuários tentaram clicar várias vezes sobre o nome do docente, turma e até sala, mas também não obtiveram resposta do sistema, já que esses elementos não são links, nem representam botões de ação.

4.2.7.2.7 *Fat Interface*

Esse *smell* não pôde ser corretamente detectado devido à um componente usado pelo sistema que faz a geração automática dos IDs dos elementos da página, quando ela é carregada. Esse componente, que é usado em parte do sistema, faz com que um mesmo elemento possa ter IDs diferentes cada vez que é carregado. Assim, como o ID do elemento faz parte do seu XPath, e o XPath é utilizado pela abordagem para identificá-lo, um mesmo elemento pode ser contado como vários diferente em uma determinada página. Assim, os relatórios do *Fat Interface* chegaram a apontar milhares de elementos diferentes em uma mesma página, quando na verdade eram apenas os mesmos elementos com IDs diferentes.

4.2.8 Discussões

De acordo com os resultados obtidos, pode-se afirmar que a resposta para a QP1 é afirmativa: a quantidade de ações que podem ser analisadas com auxílio da abordagem está limitada à cerca de 800 mil ações simultaneamente. Conforme discutido na subseção 4.2.7, essa é uma limitação da forma de implementação utilizada. Ela pode ser reduzida em versões futuras da ferramenta, contudo, isso requer um estudo cuidadoso da implementação atual. Em adição, é possível analisar quantidades maiores de ações com a implementação atual, dividindo-se o processo em múltiplas detecções, com diferentes janelas temporais.

Para a QP2, a resposta também é afirmativa: a quantidade de sessões que podem ser analisadas simultaneamente com auxílio da abordagem proposta é limitada pelo poder de processamento computacional e pela quantidade de memória RAM utilizados. O limite

exato de quantas sessões é possível analisar não foi determinado com exatidão devido ao grande tempo de processamento necessário para se concluir cada teste nesse tipo de situação. Contudo, tendo em conta os testes que foram concluídos com êxito, estima-se que, com as configurações utilizadas para teste nesta avaliação, esse limite sejam de cerca de 12 a 15 mil sessões, com um tempo de processamento aproximado de 30 minutos. Também é possível, caso os recursos computacionais sejam muito limitados, dividir o processo em múltiplas detecções a fim de cobrir todos os dados capturados.

Finalmente, para a QP3, a resposta é novamente afirmativa: a abordagem pode ser utilizada para analisar sistemas de grande porte, contudo, restrições bem impactantes precisam ser estabelecidas para tornar isso factível. Apesar disso, verificamos que algumas das restrições podem ser incorporadas naturalmente à versões futuras da ferramenta sem prejuízos à detecção de *smells*. A indexação do BD, por exemplo, poder ser alterada para ocorrer automaticamente quando necessária e a limitação da quantidade de ações pode ser aliviada com a melhora da implementação. Algumas das restrições utilizadas, como a limitação das ações iniciais/finais, contudo, reduzem a precisão dos resultado. Assim, não recomenda-se utilizá-las em análises cujo objetivo principal seja avaliar a usabilidade do sistema.

Além da resposta às questões de pesquisa, foram identificados outros pontos interessantes nesta avaliação que merecem ser discutidos. O primeiro é em relação ao *smell Fat Interface*. Através dos resultados obtidos nas duas avaliações realizadas, pôde-se observar que a detecção desse *smell* enfrentou problemas relacionados aos componentes usados na implementação em ambos os casos. Assim, uma ideia que pode ser utilizada para tentar diminuir esse problema em versões futuras da ferramenta é adotar formas alternativas de identificar uma ação, já que o XPath nem sempre oferece a forma mais precisa de ser fazer isso. O conteúdo (texto) e posicionamento do elemento são fatores interessantes a se considerar. Em adição, também é necessário estabelecer uma forma de identificar interfaces que considere mais que a URL da mesma.

O desenvolvedor que participou deste estudo, no papel de avaliador, também ofereceu importante *feedback* sobre pontos que podem ser melhorados na aplicação. Primeiramente, ele disse que gostaria que a data/hora (*timestamp*) da ação também fosse mostrada no grafo de sessão. Atualmente, as únicas informações exibidas nesse grafo são URL, XPath, tipo e quantidade de ocorrências (Figura 15). Com o *timestamp* da ação seria possível identificar, por exemplo, quando o valor elevado da duração de uma sessão é causado pela perda de foco ou pausas realizadas pelo usuário.

Além disso, o avaliador também afirmou que seria interessante ver o conteúdo (texto) dos elementos no grafo de sessão. Como pôde ser visto pelos resultados desta avaliação, alguns sistemas fazem a geração automática do ID de seus elementos. Assim, nem sempre é fácil identificar o elemento pelo seu XPath quando o ID que faz parte dele é

gerado dessa forma. Outra funcionalidade que ele disse que gostaria de ver é o passo a passo das ações realizadas no grafo de sessão. Principalmente quando se tem uma grande quantidade de ações, a organização sequencial das ações nesse grafo pode se tornar confusa e difícil de visualizar, então um passo a passo executado pela própria ferramenta ajudaria a reduzir essa confusão.

4.3 Considerações Finais

Este capítulo apresentou as avaliações realizadas para analisar o desempenho da EUSADUS. Inicialmente foi detalhado como essas avaliações foram organizadas, ou seja, suas configurações, e, por fim, mostraram-se os resultados obtidos pela análise dos relatórios gerados pela ferramenta. Apesar de ser eficaz na detecção de alguns problemas, nota-se que a ferramenta precisa ser ajustada para melhorar seu desempenho, apresentar melhor os resultados e se adaptar às particularidades de alguns tipos de sistema. O próximo capítulo apresenta as conclusões obtidas até essa etapa deste trabalho e as atividades que podem ser realizadas em trabalhos futuros.

Considerações Finais

Este trabalho apresentou uma abordagem para auxiliar a realização de testes de usabilidade explorando o conceito de *usability smells*. Apesar do teste tradicional ser considerado uma das formas mais populares de avaliação, ele ainda sofre com problemas relativos à complexidade de organização e alto custo financeiro e temporal. Assim, sua execução é muitas vezes ignorada pelos desenvolvedores de aplicações Web, que acabam por prejudicar os usuários e até, possivelmente, comprometer o desempenho da aplicação.

Para alcançar os objetivos apresentados foi realizado um levantamento bibliográfico sobre o estado da arte em relação aos métodos de avaliação de usabilidade e *usability smells*, que serviu de base para fundamentar a parte teórica da pesquisa. Em seguida, foram buscadas as abordagens que proveem algum tipo de automação para esses métodos, como uma forma de se utilizar de todos os avanços obtidos até agora na área, evitando a duplicação de esforços e possibilitando a contribuição com novos achados desenvolvidos e avaliados por meio desta pesquisa.

De posse das informações adquiridas pela investigação sobre o estado da arte, uma abordagem foi desenvolvida para tentar mitigar os problemas evidenciados. Esta abordagem combina o teste tradicional com a detecção de *usability smells* de forma automática para possibilitar a realização de uma avaliação de usabilidade que demanda pouco esforço, e com resultados simples de se analisar. Essa abordagem foi integrada à ferramenta Web de auxílio a avaliações de usabilidade UseSkill, como uma extensão da mesma, se aproveitando dos dados capturados por ela para possibilitar a detecção de *smells*.

Outra importante contribuição deste trabalho foi a proposição de novos *usability smells*. Esses *smells* foram definidos com base em problemas detectados em trabalhos anteriores e na experiência do autor. Outros *smells* previamente existentes também foram adaptados para serem detectados automaticamente pela abordagem.

Para verificar a eficácia da abordagem proposta e validade das contribuições apresentadas, duas avaliações foram realizadas com a sua implementação. Os resultados mostrados indicaram que ela foi capaz de auxiliar na detecção de problemas de usabilidade em aplicações reais, que passaram despercebidos por outras formas de avaliação. Contudo, a implementação realizada ainda requer ajustes para melhorar seu desempenho e se aproximar mais dos objetivos desta pesquisa. Além disso, mais avaliações são necessárias para confirmar a legitimidade dos resultados.

É importante apontar que esta pesquisa não têm por objetivo substituir os métodos de avaliação tradicionais, uma vez que nem todos os aspectos do teste laboratorial podem ser avaliados de forma automática. Entretanto, a automação provida pela abordagem

proposta pode complementá-las, ajudando a identificar problemas que são mais facilmente evidenciados através da análise do comportamento de uma grande quantidade de usuários. Além disso, ela também pode prover uma alternativa de melhor custo/benefício, estimulando a realização do teste.

Satisfação dos Objetivos da Pesquisa

Na seção de Objetivos foram estabelecidas uma série de metas que planejava-se atingir ao início da pesquisa. Primeiramente, criar uma abordagem simples e prática, que possa ser usada por desenvolvedores com baixo nível de experiência foi a mentalidade que guiou o desenvolvimento da implementação realizada. Contudo, pelos resultados apresentados pode-se notar que esse objetivo não foi completamente satisfeito, dadas as dificuldades do avaliador em analisar alguns tipos de relatórios e a dificuldade em operar a própria ferramenta com sistemas de grande porte. Apesar disso, foram feitas importantes descobertas durante este estudo que podem auxiliar muito à se aproximar do cenário ideal.

Em adição, não foi possível avaliar a utilização da ferramenta com desenvolvedores com pouca experiência em usabilidade. Devido à dificuldade em conseguir acesso à sistemas reais, em produção, para avaliar a abordagem e à disponibilidade de desenvolvedores para participarem do estudo não conseguiu-se organizar uma avaliação nas configurações desejadas. Assim, não se pode afirmar, com certeza, que seu uso é adequado para esse tipo de avaliador. Contudo, como o desenvolvedor participante conseguiu aprender rapidamente o conceito dos *smells* usados, tem-se uma indicação positiva nesse sentido.

Já a meta de apresentar estratégias de detecção automática para *usability smells* existentes na literatura, foi satisfeita. Os *smells Too Many Layers* e *Missing Feedback* foram propostos por Almeida et al. (2015) e Harms e Grabowski (2014), respectivamente, e não possuíam estratégias de detecção automática no domínio Web. Apesar de nem sempre estarem associados diretamente à problemas de usabilidade nas avaliações realizadas, suas definições foram corretamente adaptadas, e eles ajudaram a apontar a navegação do usuário por muitas páginas diferentes e elementos que não disparavam nenhum tipo de indicação sobre quando haviam sido corretamente clicados.

O objetivo de apresentar novos *usability smells* também foi satisfeito. Os *smells Laborious Task*, *Cyclic Task* e *Lonely Action* foram propostos com base na observação de problemas de usabilidade descobertos em aplicações reais. Em adição, uma estratégia de detecção automática também foi proposta para cada um deles, e os mesmos ajudaram a apontar problemas de usabilidade em outras aplicações Web.

Por fim, a ferramenta que implementa a abordagem foi criada, na forma de uma extensão de outra ferramenta. Contudo, como pôde ser observado pela segunda avaliação realizada, essa ferramenta ainda não está pronta para ser utilizada em larga escala, pois ter

uma grande quantidade de *logs* para análise pode prejudicar seu funcionamento. Apesar disso, importantes pontos de melhoria foram identificados durante essa avaliação, o que ajuda a ferramenta a dar mais um passo nessa direção.

Além disso, o modelo que pretende-se usar para distribuir o auxílio da ferramenta é o de serviço, ou seja, com a empresa que deseja utilizá-la pagando uma taxa periódica, de acordo com sua necessidade, para que seus desenvolvedores possam solucionar os problemas de usabilidade nas aplicações desenvolvidas pela mesma de forma mais eficiente. Apesar de esse modelo não ter sido implementado ainda, ele tem um baixo custo de aplicação para quem o contrata, já que só é necessária a ferramenta para realizar a avaliação (sem custos com equipamentos extras e alocação de usuários), e também de manutenção, uma vez que isso consiste, basicamente, em manter o servidor da aplicação em funcionamento. Ademais, o primeiro passo para transformar essa ideia em realidade já foi dado, tendo em conta que a ferramenta já está sendo executada como uma aplicação Web.

Trabalhos Futuros

Os principais trabalhos futuros que precisam ser desenvolvidos são as implementações dos pontos de melhoria identificados no Capítulo 4. Inicialmente, a ferramenta precisa ser refatorada para eliminar a limitação no número de ações que podem ser analisadas simultaneamente, já que esse limite se deu por conta de problemas na implementação. Além disso, outras melhorias como a idexação automática do Banco de Dados e a otimização da própria ferramenta devem ser implementadas para que a mesma possa ser utilizada em larga escala sem prejuízos para a execução das avaliações.

Também foram identificados pontos de melhoria na interface. Através dos comentários do avaliador, percebeu-se que mais informações precisam ser mostradas nos grafos de sessão para facilitar a identificação das ações e do comportamento do usuário. O passo a passo automático das ações no grafo também facilitaria a identificação do caminho percorrido. Além disso, alguns dos *smells* utilizados precisam de ajustes em sua estratégia de detecção para reduzir os problemas ocasionados por elementos particulares da implementação de alguns sistemas.

Outro ponto que pode ser melhorado é a visualização dos resultados. A maior parte dos relatórios apresentados atualmente é textual. Apesar de ser possível identificar as ações pelas informações apresentadas, essa identificação ainda tem que ser feita de forma manual, com o avaliador procurando pelo XPath dos elementos no código fonte das páginas indicadas. Uma forma de melhorar essa visualização é redirecionar o avaliador para a página correspondente, ao clicar na ação, e automaticamente buscar e destacar o elemento afetado.

Além dos pontos identificados, pretende-se realizar a incorporação de novos *smells*

na ferramenta. Conforme mostrado no Capítulo 2 existem diversos smells apresentados em outros trabalhos que podem ser adaptados para o contexto da ferramenta. Em adição, novos *smells* também podem ser propostos de acordo com a experiência adquirida através da realização desta pesquisa e em problemas de usabilidade observados nas avaliações de outros trabalhos relacionados.

Após todas as melhorias serem implementadas, novas avaliações serão realizadas com a abordagem para validar as alterações realizadas e verificar se não foram introduzidos novos problemas. Além disso, a abordagem proposta também precisa ser comparada com outros métodos de avaliação tradicional, para entender as diferenças entre eles, e aplicada a mais sistemas, para analisar a capacidade de generalização dos resultados já obtidos.

Com base nos trabalhos relacionados e na relevância em relação ao tema desta pesquisa, estabeleceu-se o local onde se tentará conseguir a publicação deste estudo. Assim, a versão resumida deste trabalho foi submetida ao 34º *ACM/SIGAPP Symposium On Applied Computing* (SAC 2019), consolidando a fase atual da pesquisa.

Referências

AHMAD, W. F. W.; SULAIMAN, S.; JOHARI, F. S. Usability Management System (USEMATE): A web-based automated system for managing usability testing systematically. In: *2010 International Conference on User Science and Engineering (i-USEr)*. Piscataway, NJ, EUA: IEEE, 2010. p. 110–115. Citado na página 3.

ALMEIDA, D. et al. Towards a catalog of usability smells. In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. Nova Iorque, NY, EUA: ACM, 2015. (SAC '15), p. 175–181. ISBN 978-1-4503-3196-8. Disponível em: <http://doi.acm.org/10.1145/2695664.2695670>. Citado 8 vezes nas páginas 25, 27, 32, 38, 42, 58, 82 e 93.

ALMEIDA, D. F. d. C. *Catálogo de usability smells*. Dissertação (Mestrado) — Universidade do Minho, 2014. Citado 4 vezes nas páginas 3, 13, 14 e 19.

ALONSO-RíOS, D. et al. An HTML analyzer for the study of web usability. In: *2009 IEEE International Conference on Systems, Man and Cybernetics*. Piscataway, NJ, EUA: IEEE, 2009. p. 1224–1229. ISSN 1062-922X. Citado na página 46.

BALBO, S. et al. Leading web usability evaluations to WAUTER. *Proceedings of the 11th AusWeb, Gold Coast, Australia*, 2005. Citado 2 vezes nas páginas 23 e 24.

BASILI, V. R. *Software Modeling and Measurement: The Goal/Question/Metric Paradigm*. College Park, MD, EUA, 1992. Citado 2 vezes nas páginas 70 e 71.

BOWMAN, D. A.; GABBARD, J. L.; HIX, D. A survey of usability evaluation in virtual environments: classification and comparison of methods. *Presence: Teleoperators and Virtual Environments*, MIT Press, v. 11, n. 4, p. 404–424, 2002. Citado na página 9.

BRINCK, T.; HOFER, E. Automatically evaluating the usability of web sites. In: *CHI '02 Extended Abstracts on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 2002. (CHI EA '02), p. 906–907. ISBN 1-58113-454-1. Disponível em: <http://doi.acm.org/10.1145/506443.506652>. Citado na página 11.

BRUUN, A. et al. Let your users do the testing: A comparison of three remote asynchronous usability testing methods. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 2009. (CHI '09), p. 1619–1628. ISBN 978-1-60558-246-7. Disponível em: <http://doi.acm.org/10.1145/1518701.1518948>. Citado na página 11.

BURZACCA, P.; PATERNò, F. Remote usability evaluation of mobile web applications. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 8004 LNCS, n. PART 1, p. 241–248, 2013. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84880689691&doi=10.1007%2f978-3-642-39232-0_27&partnerID=40&md5=70d694f014b8b59b873afb43a488b402>. Citado na página 18.

CASSINO, R.; TUCCI, M. Developing usable web interfaces with the aid of automatic verification of their formal specification. *Journal of Visual Languages & Computing*, Elsevier, v. 22, n. 2, p. 140–149, 2011. Citado na página 1.

CASSINO, R. et al. Empirical validation of an automatic usability evaluation method. *Journal of Visual Languages & Computing*, Elsevier, v. 28, p. 1–22, 2015. Citado na página 1.

CASTILLO, J. C.; HARTSON, H. R.; HIX, D. Remote usability evaluation: Can users report their own critical incidents? In: *CHI 98 Conference Summary on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 1998. (CHI '98), p. 253–254. ISBN 1-58113-028-7. Disponível em: <http://doi.acm.org/10.1145/286498.286736>. Citado na página 13.

DAMEVSKI, K. et al. Mining sequences of developer interactions in visual studio for usage smells. *IEEE Transactions on Software Engineering*, Institute of Electrical and Electronics Engineers Inc., v. 43, n. 4, p. 359–371, 2017. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85018977446&doi=10.1109%2fTSE.2016.2592905&partnerID=40&md5=b0fbdc5052c12dba5a5b044b5ac0f506>. Citado 4 vezes nas páginas 22, 23, 58 e 93.

DISTANTE, D. et al. Business processes refactoring to improve usability in e-commerce applications. *Electronic Commerce Research*, Springer New York LLC, v. 14, n. 4, p. 497–529, 2014. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84946240935&doi=10.1007%2fs10660-014-9149-0&partnerID=40&md5=f8e88d1c785a8be094b996f424cc3e20>. Citado na página 93.

FERNANDEZ, A.; INSFRAN, E.; ABRAHÃO, S. Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology*, Elsevier, v. 53, n. 8, p. 789–817, 2011. Citado 7 vezes nas páginas 1, 2, 3, 7, 8, 9 e 10.

FERRÉ, X. et al. Usability basics for software developers. *IEEE software*, IEEE, v. 18, n. 1, p. 22–29, 2001. Citado 2 vezes nas páginas 23 e 24.

FILHO, I. B.; AQUINO, G.; ROSA, J. G. S. SIGAA mobile – o caso de sucesso da ferramenta de gestão acadêmica na era da computação móvel. In: *Anais do XXIV Simpósio Brasileiro de Informática na Educação*. Porto Alegre, RS, Brasil: Sociedade Brasileira de Computação, 2013. Disponível em: <https://doi.org/10.5753%2Fcbie.sbie.2013.92>. Citado na página 71.

FOWLER, M.; BECK, K. *Refactoring: improving the design of existing code*. Boston, MA, EUA: Addison-Wesley Professional, 1999. Citado 3 vezes nas páginas 3, 13 e 25.

GARRETT, J. J. Ajax: A new approach to web applications. Adaptive Path, 2005. Citado na página 43.

GARRIDO, A. et al. Data-driven usability refactoring: Tools and challenges. In: *2017 6th International Workshop on Software Mining (SoftwareMining)*. Piscataway, NJ, EUA: IEEE, 2017. p. 52–55. Citado na página 93.

GARRIDO, A.; ROSSI, G.; DISTANTE, D. Model refactoring in web applications. In: *Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution*. Washington, DC, EUA: IEEE Computer Society, 2007. (WSE '07), p. 89–96. ISBN

978-1-4244-1450-5. Disponível em: <https://doi.org/10.1109/WSE.2007.4380249>. Citado na página 19.

GARRIDO, A.; ROSSI, G.; DISTANTE, D. Refactoring for usability in web applications. *IEEE Softw.*, IEEE Computer Society Press, Los Alamitos, CA, EUA, v. 28, n. 3, p. 60–67, maio 2011. ISSN 0740-7459. Disponível em: <https://doi.org/10.1109/MS.2010.114>. Citado 4 vezes nas páginas 3, 14, 19 e 93.

GRIGERA, J. et al. Assessing refactorings for usability in e-commerce applications. *Empirical Softw. Engg.*, Kluwer Academic Publishers, Hingham, MA, EUA, v. 21, n. 3, p. 1224–1271, jun. 2016. ISSN 1382-3256. Disponível em: <http://dx.doi.org/10.1007/s10664-015-9384-6>. Citado na página 93.

GRIGERA, J.; GARRIDO, A.; RIVERO, J. M. A tool for detecting bad usability smells in an automatic way. In: CASTELEYN, S.; ROSSI, G.; WINCKLER, M. (Ed.). *Web Engineering*. Cham, Suíça: Springer International Publishing, 2014. p. 490–493. ISBN 978-3-319-08245-5. Citado 3 vezes nas páginas 3, 17 e 93.

GRIGERA, J. et al. Automatic detection of usability smells in web applications. *International Journal of Human-Computer Studies*, v. 97, p. 129 – 148, 2017. ISSN 1071-5819. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1071581916301215>. Citado 16 vezes nas páginas 9, 2, 3, 12, 13, 14, 17, 29, 32, 38, 43, 44, 58, 62, 73 e 93.

GRIGERA, J.; GARRIDO, A.; ROSSI, G. Kobold: Web usability as a service. In: *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*. Piscataway, NJ, EUA: IEEE Press, 2017. (ASE 2017), p. 990–995. ISBN 978-1-5386-2684-9. Disponível em: <http://dl.acm.org/citation.cfm?id=3155562.3155692>. Citado na página 93.

HARMS, P.; GRABOWSKI, J. Usage-based automatic detection of usability smells. In: *Proceedings of the 5th IFIP WG 13.2 International Conference on Human-Centered Software Engineering - Volume 8742*. Berlim, Heidelberg, Alemanha: Springer-Verlag, 2014. (HCSE 2014), p. 217–234. ISBN 978-3-662-44810-6. Disponível em: <https://doi.org/10.1007/978-3-662-44811-3_13>. Citado 12 vezes nas páginas 7, 13, 14, 23, 24, 32, 38, 44, 45, 58, 82 e 93.

HERBOLD, S.; HARMS, P. AutoQUEST – Automated Quality Engineering of Event-Driven Software. In: *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. Washington, DC, EUA: IEEE Computer Society, 2013. (ICSTW '13), p. 134–139. ISBN 978-0-7695-4993-4. Disponível em: <http://dx.doi.org/10.1109/ICSTW.2013.23>. Citado na página 24.

HERMANS, F.; PINZGER, M.; DEURSEN, A. v. Detecting and visualizing inter-worksheet smells in spreadsheets. In: *Proceedings of the 34th International Conference on Software Engineering*. Piscataway, NJ, EUA: IEEE Press, 2012. (ICSE '12), p. 441–451. ISBN 978-1-4673-1067-3. Disponível em: <http://dl.acm.org/citation.cfm?id=2337223.2337275>. Citado na página 25.

HONG, J. I. et al. WebQuilt: A proxy-based approach to remote web usability testing. *ACM Transactions on Information Systems*, v. 19, n. 3, p. 263–285, 2001. Citado 2 vezes nas páginas 9 e 10.

ISO. *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – part 11: Guidance on usability*. Genebra, Suíça, 1998. Citado na página 7.

IVORY, M. Y.; HEARST, M. A. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, ACM, Nova Iorque, NY, EUA, v. 33, n. 4, p. 470–516, dez. 2001. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/503112.503114>. Citado 7 vezes nas páginas 1, 5, 7, 8, 9, 11 e 12.

JAAFAR, A. bt et al. e-RUE: A cheap possible solution for usability evaluation. In: IEEE. *Information Technology, 2008. ITSim 2008. International Symposium on*. Piscataway, NJ, EUA: IEEE, 2008. v. 3, p. 1–5. Citado 2 vezes nas páginas 1 e 2.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Durham, Reino Unido, 2007. Citado na página 18.

KIURA, M.; OHIRA, M.; MATSUMOTO, K.-I. Webjig: An automated user data collection system for website usability evaluation. In: *Proceedings of the 13th International Conference on Human-Computer Interaction. Part I: New Trends*. Berlim, Heidelberg, Alemanha: Springer-Verlag, 2009. p. 277–286. ISBN 978-3-642-02573-0. Disponível em: <http://dx.doi.org/10.1007/978-3-642-02574-7_31>. Citado na página 2.

KUZNIARZ, L.; STARON, M.; WOHLIN, C. Students as study subjects in software engineering experimentation. 2003. Citado na página 59.

LECEROF, A.; PATERNò, F. Automatic support for usability evaluation. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, EUA, v. 24, n. 10, p. 863–888, out. 1998. ISSN 0098-5589. Disponível em: <https://doi.org/10.1109/32.729686>. Citado na página 24.

LI, C.-h.; KIT, C.-c. Web structure mining for usability analysis. In: *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web Intelligence*. Washington, DC, EUA: IEEE Computer Society, 2005. (WI '05), p. 309–312. ISBN 0-7695-2415-X. Disponível em: <http://dx.doi.org/10.1109/WI.2005.160>. Citado na página 3.

MARTIN, R. C.; MARTIN, M. *Agile Principles, Patterns, and Practices in C# (Robert C. Martin)*. Upper Saddle River, NJ, EUA: Prentice Hall PTR, 2006. ISBN 0131857258. Citado na página 45.

MATERA, M.; RIZZO, F.; CARUGHI, G. T. Web usability: Principles and evaluation methods. In: MENDES, E.; MOSLEY, N. (Ed.). *Web Engineering*. Berlim, Heidelberg, Alemanha: Springer Berlin Heidelberg, 2006. p. 143–180. ISBN 978-3-540-28218-1. Disponível em: <https://doi.org/10.1007/3-540-28218-1_5>. Citado na página 8.

MEDINA, N. M. et al. An incremental approach for building accessible and usable web applications. In: *Proceedings of the 11th International Conference on Web Information Systems Engineering*. Berlim, Heidelberg, Alemanha: Springer-Verlag, 2010. (WISE'10), p. 564–577. ISBN 3-642-17615-1, 978-3-642-17615-9. Disponível em: <http://dl.acm.org/citation.cfm?id=1991336.1991397>. Citado na página 19.

MOBASHER, B. Web usage mining. In: *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data*. Berlim, Heidelberg, Alemanha: Springer Berlin Heidelberg, 2007. p. 449–483. ISBN 978-3-540-37882-2. Disponível em: <https://doi.org/10.1007/978-3-540-37882-2_12>. Citado na página 20.

MOSQUEIRA-REY, E. et al. A multi-agent system based on evolutionary learning for the usability analysis of websites. In: NGUYEN, N. T.; JAIN, L. C. (Ed.). *Intelligent Agents in the Evolution of Web and Applications*. Berlim, Heidelberg, Alemanha: Springer Berlin Heidelberg, 2009. p. 11–34. ISBN 978-3-540-88071-4. Disponível em: <https://doi.org/10.1007/978-3-540-88071-4_2>. Citado na página 9.

NEBELING, M.; SPEICHER, M.; NORRIE, M. C. Crowdstudy: General toolkit for crowdsourced evaluation of web interfaces. In: *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Nova Iorque, NY, EUA: ACM, 2013. (EICS '13), p. 255–264. ISBN 978-1-4503-2138-9. Disponível em: <http://doi.acm.org/10.1145/2494603.2480303>. Citado na página 18.

NIELSEN, J. *Usability Engineering*. São Francisco, CA, EUA: Morgan Kaufmann Publishers Inc., 1993. ISBN 0125184050. Citado 4 vezes nas páginas 42, 43, 46 e 67.

NIELSEN, J. Usability inspection methods. In: *Conference Companion on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 1994. (CHI '94), p. 413–414. ISBN 0-89791-651-4. Disponível em: <http://doi.acm.org/10.1145/259963.260531>. Citado na página 9.

NORMAN, D. A. *The Design of Everyday Things*. Nova Iorque, NY, EUA: Basic Books, Inc., 2002. ISBN 9780465067107. Citado 2 vezes nas páginas 23 e 24.

OLSINA, L. et al. Web application evaluation and refactoring: a quality-oriented improvement approach. *Journal of Web Engineering*, Rinton Press Inc., v. 7, n. 4, p. 258–280, 2008. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-63149147247&partnerID=40&md5=33f77bba1135566b12946ac10d764b8e>. Citado na página 19.

OLSINA, L. et al. Incremental quality improvement in web applications using web model refactoring. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 4832 LNCS, p. 411–422, 2007. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-38149045783&partnerID=40&md5=e75c56240225c2e27fe2d8ae7c85d9ac>. Citado na página 19.

OPDYKE, W. F. *Refactoring object-oriented frameworks*. Tese (Doutorado) — University of Illinois at Urbana-Champaign, 1992. Citado na página 3.

PATERNÒ, F.; PIRUZZA, A.; SANTORO, C. Remote usability analysis of multimodal information regarding user behaviour. In: *International COST 294 Workshop on User Interface Quality Models*. Berlim, Heidelberg, Alemanha: Springer-Verlag, 2005. p. 15. Citado na página 24.

PATERNò, F.; SCHIAVONE, A.; CONTE, A. Customizable automatic detection of bad usability smells in mobile accessed web applications. In: . Association for Computing Machinery, Inc, 2017. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85030329906&doi=10.1145%2f3098279.3098558&partnerID=40&md5=90ca3e5adabd0261f0ba6edfc6544705>. Citado 3 vezes nas páginas 27, 58 e 93.

PAUL, A.; YADAMSUREN, B.; ERDELEZ, S. An experience with measuring multi-user online task performance. In: *2012 World Congress on Information and Communication Technologies*. Piscataway, NJ, EUA: IEEE, 2012. p. 639–644. Citado na página 1.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, Reino Unido: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77. Disponível em: <http://dl.acm.org/citation.cfm?id=2227115.2227123>. Citado na página 18.

POLSON, P. G. et al. Cognitive walkthroughs: a method for theory-based evaluation of user interfaces. *International Journal of man-machine studies*, Elsevier, v. 36, n. 5, p. 741–773, 1992. Citado 2 vezes nas páginas 23 e 24.

POUR, P. A.; CALVO, R. Towards a generic framework for automatic measurements of web usability using affective computing techniques. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, v. 6974 LNCS, n. PART 1, p. 447–456, 2011. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-80054835633&doi=10.1007%2f978-3-642-24600-5_48&partnerID=40&md5=2fb512160851a391f5e3dcc4b9dc08ac>. Citado na página 18.

SANTANA, V. F. de; BARANAUSKAS, M. C. C. WELFIT: A remote evaluation tool for identifying web usage patterns through client-side logging. *International Journal of Human-Computer Studies*, Elsevier, v. 76, p. 40–49, 2015. Citado 2 vezes nas páginas 12 e 18.

SCHOLTZ, J. Adaptation of traditional usability testing methods for remote testing. In: *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*. Piscataway, NJ, EUA: IEEE, 2001. p. 8 pp.–. Citado na página 2.

SILVA, J. a. C. et al. The GUISurfer tool: Towards a language independent approach to reverse engineering GUI code. In: *Proceedings of the 2Nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. Nova Iorque, NY, EUA: ACM, 2010. (EICS '10), p. 181–186. ISBN 978-1-4503-0083-4. Disponível em: <http://doi.acm.org/10.1145/1822018.1822045>. Citado na página 27.

SOMEREN, M. v. et al. *The think aloud method: a practical approach to modelling cognitive processes*. Amsterdã, Países Baixos: Academic Press, 1994. Citado na página 9.

SOUSA, L. et al. How do software developers identify design problems?: A qualitative analysis. In: *Proceedings of the 31st Brazilian Symposium on Software Engineering*. Nova Iorque, NY, EUA: ACM, 2017. (SBES'17), p. 54–63. ISBN 978-1-4503-5326-7. Disponível em: <http://doi.acm.org/10.1145/3131151.3131168>. Citado 2 vezes nas páginas 38 e 45.

SOUZA, M. et al. Uma abordagem para apoiar avaliações de usabilidade em sistemas web com base em mineração de dados. *XV Simpósio Brasileiro de Qualidade de Software (SBQS)*, 2016. Citado 15 vezes nas páginas 9, 19, 20, 22, 33, 38, 39, 47, 58, 60, 62, 63, 67, 68 e 69.

SOUZA, M. et al. Useskill: uma ferramenta que auxilia na realização de avaliações de usabilidade em aplicações web. *VII Congresso Brasileiro de Software: Teoria e Prática (CBSoft)*, 2016. Citado na página 19.

SOUZA, M. M. C. *Uma Abordagem para Apoiar Avaliações de Usabilidade de Sistemas Web Remotamente*. Dissertação (Mestrado) — Universidade Federal do Piauí, 2016. Citado 3 vezes nas páginas 17, 32 e 48.

SOUZA, M. M. C. et al. UseSkill: uma ferramenta de apoio à avaliação de usabilidade de sistemas web. *XIV Simpósio Brasileiro de Qualidade de Software (SBQS)*, 2015. Citado 2 vezes nas páginas 18 e 19.

SPOOL, J.; SCHROEDER, W. Testing web sites: Five users is nowhere near enough. In: *CHI '01 Extended Abstracts on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 2001. (CHI EA '01), p. 285–286. ISBN 1-58113-340-5. Disponível em: <http://doi.acm.org/10.1145/634067.634236>. Citado na página 10.

SZCZUR, M. R.; SHEPPARD, S. B. TAE Plus: Transportable Applications Environment Plus: A user interface development environment. *ACM Trans. Inf. Syst.*, ACM, Nova Iorque, NY, EUA, v. 11, n. 1, p. 76–101, jan. 1993. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/151480.151509>. Citado na página 18.

TIDWELL, J. *Designing interfaces: Patterns for effective interaction design*. Sebastopol, CA, EUA: O'Reilly Media, Inc., 2010. Citado 3 vezes nas páginas 9, 24 e 46.

TRAVIS, D. *A Business Case for Usability*. Londres, Reino Unido: Userfocus, 2007. Disponível em: <https://www.userfocus.co.uk/articles/usabilitybenefits.html>. Acesso em: 7 ago. 2018. Citado na página 3.

TULLIS, T. et al. An empirical comparison of lab and remote usability testing of websites. In: *Proceedings of Usability Professionals Conference*. Orlando, FL, EUA: Usability Professionals' Association, 2002. Citado na página 2.

UEHLING, D. L.; WOLF, K. User Action Graphing Effort (UsAGE). In: *Conference Companion on Human Factors in Computing Systems*. Nova Iorque, NY, EUA: ACM, 1995. (CHI '95), p. 290–291. ISBN 0-89791-755-3. Disponível em: <http://doi.acm.org/10.1145/223355.223679>. Citado na página 18.

VARGAS, A.; WEFFERS, H.; ROCHA, H. V. D. Discovering and analyzing patterns of usage to detect usability problems in web applications. In: *Proceedings of the 2011 11th International Conference on Intelligent Systems Design and Applications*. Piscataway, NJ, EUA: IEEE, 2011. p. 575 – 580. ISSN 21647143. Disponível em: <http://dx.doi.org/10.1109/ISDA.2011.6121717>. Citado 2 vezes nas páginas 18 e 19.

VARGAS, A.; WEFFERS, H.; ROCHA, H. V. da. A method for remote and semi-automatic usability evaluation of web-based applications through users behavior analysis. In: *Proceedings of the 7th International Conference on Methods and Techniques in Behavioral Research*. Nova Iorque, NY, EUA: ACM, 2010. (MB '10), p. 19:1–19:5. ISBN 978-1-60558-926-8. Disponível em: <http://doi.acm.org/10.1145/1931344.1931363>. Citado 3 vezes nas páginas 2, 18 e 19.

VASCONCELOS, L. G. de; BALDOCHI JR., L. A. Towards an automatic evaluation of web applications. In: *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Nova Iorque, NY, EUA: ACM, 2012. (SAC '12), p. 709–716. ISBN 978-1-4503-0857-1. Disponível em: <http://doi.acm.org/10.1145/2245276.2245410>. Citado 2 vezes nas páginas 2 e 18.

W3C. *XML Path Language (XPath)*. Cambridge, MA, EUA, 1999. Citado na página 34.

WOHLIN, C. et al. *Experimentation in Software Engineering*. Nova Iorque, NY, EUA: Springer Publishing Company, Incorporated, 2012. ISBN 3642290434, 9783642290435. Citado 2 vezes nas páginas 59 e 60.

XU, L.; XU, B. Applying agent into intelligent web application testing. In: *Proceedings of the 2007 International Conference on Cyberworlds*. Washington, DC, EUA: IEEE Computer Society, 2007. (CW '07), p. 61–65. ISBN 0-7695-3005-2. Disponível em: <https://doi.org/10.1109/CW.2007.16>. Citado na página 18.

APÊNDICE A – Trabalhos Identificados na Pesquisa Sistemática

Tabela 5 – Trabalhos relacionados à detecção de *usability smells*.

Título	**Autores**
Automatic detection of usability smells in web applications	Grigera et al. (2017)
Kobold: Web usability as a service	Grigera, Garrido e Rossi (2017)
Data-driven usability refactoring: Tools and challenges	Garrido et al. (2017)
Assessing refactorings for usability in e-commerce applications	Grigera et al. (2016)
A tool for detecting bad usability smells in an automatic way	Grigera, Garrido e Rivero (2014)
Business processes refactoring to improve usability in E-commerce applications	Distante et al. (2014)
Refactoring for usability in web applications	Garrido, Rossi e Distante (2011)
Customizable automatic detection of bad usability smells in mobile accessed web applications	Paternò, Schiavone e Conte (2017)
Mining Sequences of Developer Interactions in Visual Studio for Usage Smells	Damevski et al. (2017)
Towards a catalog of usability smells	Almeida et al. (2015)
Usage-based automatic detection of usability smells	Harms e Grabowski (2014)

"Descoberta de Problemas de Usabilidade em Aplicações Web a partir da Detecção Automática de Usability Smells"

RAFAEL FONTINELE RIBEIRO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Centro de Ciências da Natureza da Universidade Federal do Piauí, como parte integrante dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Aprovada por:

Prof. Pedro de Alcântara dos Santos Neto
(Presidente da Banca Examinadora)

Prof. Lincoln Souza Rocha
(Examinador Externo)

Prof. Raimundo Santos Moura
(Examinador Interno)

Teresina, 29 de agosto de 2018