A study based on TAM model about debugging tools

Fabio Pereira da Silva
University of São Paulo
fabio.dasilva@alumni.usp.br

ABSTRACT

Debugging is the task of locating and fixing defects in a program. Despite the increase of its importance in last decades, debugging is responsible for a large part of costs in a software project by organizations. Among the techniques proposed to minimize these difficulties, Spectrum-Based Fault Localization (SFL) is a promising debugging technique due to it is relative low execution cost. Recently, visualization tools have been proposed to represent the suspicious values of program elements with SFL techniques in different metaphors. Some tools use textual representation and others a visual representation. In this paper, we compare two SFL debugging tools. Jaguar presents the most suspicious elements of a program in a list sorted by suspicious values. CodeForest represents the program in a three dimensional cacti forest. In this article are presented the results of an evaluation with 119 students to assess the usability perception of these tools to the fault localization by Technology Acceptance Model (TAM). This model aims to help organizations during the evaluation of new technologies. The results of study show that Jaguar has greater usability than CodeForest; however, the statistical effect size observed is low between them.

KEYWORDS

TAM Model; Experimental Software Engineering; Quantitative Analysis

1 INTRODUÇÃO

Para Akari, Furukawa e Cheng [2], depuração consiste na localização e correção de defeitos presentes em um sistema. Teste de software não é depuração. A depuração é uma consequência do teste bem-sucedido, ou seja, o teste que conseguiu identificar a presença de defeitos [2]. Esse teste é avaliado até que uma divergência entre o resultado esperado e os valores reais obtidos seja encontrada [2].

As atividades de teste de software e depuração demandam uma parcela significativa de tempo durante o processo de desenvolvimento de software e correspondem até 75% dos custos ao longo do processo evolutivo de um sistema, causando grande prejuízo as organizações [9]. Entretanto, mesmo com o crescimento de estudos que visem trazer melhorias para a realização desta atividade apresentados nos últimos anos, a atividade de depuração ainda é realizada praticamente da mesma forma desde a década de 1960 quando os primeiros depuradores foram introduzidos.

Dentre as técnicas propostas nos últimos anos para a localização de defeitos, o fatiamento de programas realiza a seleção de um conjunto de comandos que afetam os valores de uma ou mais variáveis em determinado ponto do programa, visando que o desenvolvedor concentre a sua atenção em determinado trecho do código [22]. A depuração delta identifica trechos do programa que provocaram uma falha comparando o estado da aplicação em situações em que a falha ocorre com outras em que ela não é apresentada [23].

Entretanto, o fatiamento de programas gera uma quantidade excessiva de comandos suspeitos e a depuração delta consome muito tempo e memória para obter seus resultados. Ou seja, estas técnicas não são escaláveis para utilização em situações reais de desenvolvimento [16].

Para Renieris e Reiss [16], a técnica de localização de defeitos baseada em cobertura, também conhecida como LDC, é definida por um conjunto de componentes (comandos, blocos, predicados, associações definição-uso ou subprogramas) cobertos durante a execução de um teste. LDC é uma técnica de depuração que utiliza dados de cobertura dos casos de teste para verificar quais componentes possuem potencial de conter um defeito [16]. Esse potencial é calculado a partir de métricas, como Ochiai [1] e Tarântula [10].

As métricas utilizam as informações de cobertura para inferir os elementos de um programa mais suspeitos de conterem defeitos [10]. Elas são baseadas em coeficientes que levam em consideração os componentes que foram, ou não foram, executados pelos casos de testes.

Nas últimas décadas foram realizados estudos sobre o uso de ferramentas visuais de depuração que utilizam técnicas LDC para a localização de defeitos. As ferramentas Jaguar [17] e CodeForest [14] utilizam diferentes metáforas visuais para representar as informações de suspeição associadas ao código do programa.

Jaguar apresenta os elementos mais suspeitos (linhas ou adus) na forma de uma lista. CodeForest demonstra os elementos mais suspeitos de um programa em uma floresta de cactus tridimensional.

Foram encontrados poucos trabalhos na literatura que realizem comparações sobre o uso de ferramentas visuais e textuais na atividade de depuração que utilizem técnicas LDC para a localização de defeitos [15]. O autor deste artigo apresentou os resultados preliminares da avaliação com as ferramentas CodeForest e Jaguar [7]. Gouveia, Campos e Abreu [9] realizaram um estudo com a ferramenta GZoltar. Entretanto o estudo foi bastante restrito na comparação com depuradores simbólicos e não realizou nenhuma avaliação sobre a usabilidade de ferramentas visuais. Com base nesta situação é possível perceber a necessidade do desenvolvimento e avaliação de novas ferramentas que auxiliem os desenvolvedores na localização de defeitos.

Este artigo apresenta os resultados de um estudo com as ferramentas CodeForest e Jagar com 119 estudantes para avaliar a percepção de usabilidade das ferramentas por meio do modelo *Technology Acceptance Model* (TAM). Com a aplicação do modelo buscou-se identificar como as ferramentas podem ser incluídas no processo de transformação digital das organizações de desenvolvimento de software, a partir da identificação de qual ferramenta possuir maior percepção de usabilidade.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta conceitos básicos sobre o modelo TAM. A Seção

3 aborda os trabalhos relacionados encontrados na literatura. Na Seção 4 são contextualizadas as ferramentas avaliadas. Em seguida, na Seção 5 é detalhado o desenho experimental aplicado no estudo e os métodos estatísticos utilizados para a análise de dados. A Seção 6 apresenta os resultados obtidos no projeto de pesquisa. A Seção 7 as ameaças à validade. Por último, a Seção 8 dedica-se as considerações finais.

2 TECHNOLOGY ACCEPTANCE MODEL (TAM)

A avaliação de novas tecnologias está sendo objetivo de estudo desde década de 1970 e os trabalhos abrangem vários fatores que podem afetar em determinado grau a utilização de uma dada tecnologia em situações reais de desenvolvimento [13]. O modelo TAM é um dos mais utilizados pelos pesquisadores para descrever a aceitação de uma dada tecnologia [21].

O modelo TAM apresentado por Davis [8] visa explicar variáveis que fazem com que os usuários aceitem ou rejeitem um sistema. O modelo proposto explica as determinantes mais afetadas para a utilização de um sistema de informação com base em uma análise indutiva de fatores.

O TAM surgiu através de um contrato entre a IBM Canadá e o *Massachusetts Institute of Technology* (MIT) na década de 1980 visando avaliar o potencial de mercado de novos produtos da marca e possibilitar uma explicação das determinantes que levavam os usuários à utilização de computadores e pode ser aplicado em diversas áreas [8].

O TAM está alicerçado em três pilares fundamentais, classificados por Davis [8]: a utilidade percebida, facilidade de uso percebida e intenção de uso futuro da tecnologia. A utilidade é dada pelo grau com que uma pessoa acredita que o sistema pode melhorar o seu desempenho. A facilidade de uso é dada como o nível em que uma pessoa acredita que o uso de um sistema será de livre esforço. A intenção de uso é mensurada através do comportamento que o usuário apresenta para a execução de uma dada tarefa futura na aplicação.

3 TRABALHOS RELACIONADOS

Na literatura foram encontradas poucas ferramentas visuais de depuração que utilizam técnicas LDC para a localização de defeitos. Dentre elas estão as ferramentas Tarantula [11] e GZoltar [9].

Jones, Harrold e Stasko [11] propuseram a ferramenta Tarantula. Ela tem como objetivo auxiliar os desenvolvedores na atividade de depuração por meio de uma ferramenta gráfica que utiliza cores para mapear visualmente o programa avaliado.

Tarantula identifica os casos de testes que foram executados com sucesso ou com falha e representa as informações a partir de um esquema de coloração. Com base nestas informações, os desenvolvedores podem inspecionar os trechos mais suspeitos do programa.

GZoltar proposta por Gouveia, Campos e Abreu [9] possui um esquema de cores que indica a probabilidade de cada elemento do programa conter defeitos. A ferramenta elabora uma lista dos elementos candidatos a conter o defeito ordenada pelo valor de suspeição de cada um deles que podem ser pacotes, classes, métodos ou linhas de um programa. As informações da aplicação podem ser visualizadas de três maneiras: representação em anel, particionamento vertical e ordenação hierárquica [9].

As visualizações geradas são interativas, permitindo que o usuário navegue na estrutura do projeto e exibidas a partir do esquema de coloração. A cor verde representa um baixo valor de suspeição, amarelo indica um valor médio, laranja representa um percentual alto e vermelho indica os elementos do programa com maior probabilidade de conter defeitos [9].

Golveia, Campos e Abreu [9] realizaram um estudo com a ferramenta GZoltar para avaliar sua eficácia e eficiência. Porém o trabalho considerou apenas uma comparação com depuradores simbólicos sem o uso de nenhuma ferramenta textual de depuração, tornando necessário a elaboração de novos estudos. Também não foi realizada nenhuma avaliação de usabilidade da ferramenta proposta.

O autor deste trabalho apresentou os resultados parciais da avaliação de usabilidade das ferramentas CodeForest e Jaguar; entretanto, o estudo havia sido conduzido com uma amostra de pessoas que ainda estavam na metade do curso de graduação. No estudo preliminar, não houve diferença estatística entre as ferramentas avaliadas [7].

4 FERRAMENTAS ESTUDADAS

4.1 Jaguar

Jaguar é uma ferramenta textual de depuração que utiliza técnicas LDC para a localização de defeitos.

Jaguar coleta informações da cobertura do código invocando testes de unidade do programa e recolhe estas informações para cada elemento avaliado. A suspeição de cada linha pode ser definida por dez métricas diferentes. A partir das informações coletadas é gerado um arquivo XML com o valor de suspeição de cada linha do programa coberta pelos testes.

A ferramenta é um plugin integrável com a IDE Eclipse. Ela permite a seleção do tipo de cobertura de código que será utilizada que pode ser de fluxo de dados ou fluxo de controle.

Jaguar recebe um objeto com todos os dados de cobertura dos casos de testes bem como a informação se um dado caso de teste falhou ou foi bem-sucedido. Com essas informações, uma das dez métricas disponíveis é utilizada para atribuir valores de suspeição a cada componente (nós ou blocos, ramos, associações definição-uso, métodos) coberto durante os testes. Esses componentes são então mapeados para linhas a serem inspecionadas pelo desenvolvedor.

Após a geração do relatório o desenvolvedor tem a opção de utilizar a JaguarView que apresenta as informações obtidas com base nos valores calculados pela Jaguar, conforme demonstrado na Figura 1.

Jaguar pode ser utilizada de duas formas, a lista de linhas e uma lista de métodos mais suspeitos. As duas versões foram consideradas no experimento.

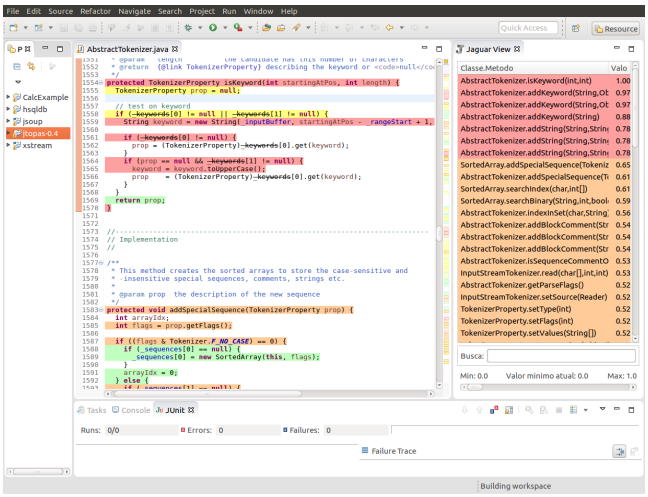

Figura 1: Representação da Jaguar

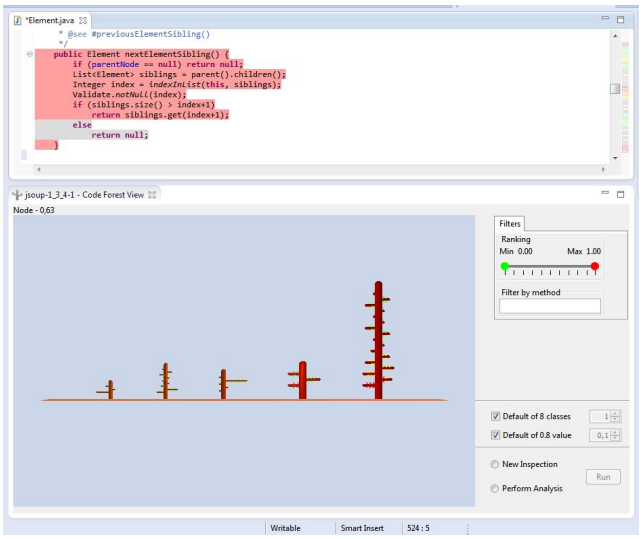

Figura 2: Representação da CodeForest

4.2 CodeForest

CodeForest tem como objetivo auxiliar os desenvolvedores na atividade de depuração por meio de representações visuais em uma floresta de cactus tridimensional.

Ela recebe listas de classes, métodos e blocos suspeitos (conjunto de instruções executadas em sequência) para construir uma floresta de cactus tridimensional representando o programa inspecionado. Na CodeForest, as classes são cactus, os métodos são galhos e os blocos de comandos avaliados são representados como espinhos de um galho. Os elementos mais suspeitos da floresta são pintados de vermelho.

O posicionamento, a coloração e a espessura dos cactus da floresta são determinados com base em sua probabilidade de conter defeitos. Os cactus são posicionados de maneira ordenada da esquerda para a direita da base onde encontram-se os elementos mais suspeitos do programa.

Os elementos da floresta são coloridos de acordo com o seu valor de suspeição, considerados no intervalo entre 0 e 1. A cor verde representa a menor probabilidade dele apresentar defeitos. A cor amarela representa um estado intermediário. A cor laranja indica uma alta probabilidade. Por último, a cor vermelha representa os elementos com maior valor de suspeição.

Na CodeForest é possível interagir diretamente com o código fonte da aplicação clicando nos cactus, galhos ou espinhos da floresta. A ferramenta pode ser configurada pelo desenvolvedor, aumentando ou diminuindo o número de cactus que ele deseja que sejam apresentados, realizar uma pesquisa textual ou um filtro de busca contendo apenas os elementos mais suspeitos. A Figura 2 apresenta a CodeForest.

5 METODOLOGIA DE PESQUISA

5.1 Desenho experimental

Para atingir o objetivo proposto, foi realizado um experimento para a avaliação da usabilidade das ferramentas CodeForest e Jaguar. O desenho experimental proposto, seguiu o mesmo modelo do experimento preliminar [7].

5.2 Ambiente do experimento

A condução do experimento ocorreu com 119 estudantes do curso de graduação em Análise e Desenvolvimento de Sistemas da Faculdade de Tecnologia da Zona Leste da cidade de São Paulo (FATEC).

A escolha da instituição para o estudo, justifica-se pelas disciplinas ministradas no curso de graduação, com foco no componente curricular de Teste de Software, totalmente relacionada ao objetivo deste trabalho.

Criou-se um ambiente nos laboratórios de informática da instituição. Esta configuração incluiu as ferramentas avaliadas neste projeto de pesquisa e materiais de treinamento sobre o uso das aplicações.

No estudo foram utilizados dois programas de código aberto desenvolvidos em Java: JSoup e XStream. O JSoup[1] é uma biblioteca Java para análise, manipulação e extração de dados HTML. O XStream[2] é uma biblioteca para a serialização de objetos XML.

O defeito no programa JSoup é um defeito real encontrado no código fonte do programa disponível no repositório. Trata-se da comparação de dois elementos utilizando *equals()* ao invés de *"=="*. O defeito encontra-se na linha *454* da classe *Element*, no método *indexInList()*.

O defeito no programa XStream foi criado no experimento realizado com a ferramenta GZoltar [9]. Trata-se de uma comparação com o operador lógico errado em uma estrutura de decisão.

[1] https://github.com/jhy/jsoup
[2] https://github.com/x-stream/xstream

Ao invés de utilizar *"targetType==null"* deveria ser utilizado *"targetType!=null"*. O defeito semeado está localizado na linha *574* da classe *AnnotationMapper*, no método *cacheConverter()*.

5.3 Procedimentos

Durante o experimento os estudantes realizaram duas tarefas com as ferramentas. Jaguar foi utilizada em duas versões, a lista de linhas e a lista de métodos mais suspeitos. Com o objetivo de garantir a aleatoriedade na distribuição dos participantes, os estudantes foram divididos em grupos conforme pode ser visualizado na Tabela 1. A alocação dos participantes ocorreu de maneira aleatória por meio do delineamento em quadrados latinos [3].

Tabela 1: Distribuição dos participantes

ID	Jaguar Método	Jaguar Linha	CodeForest
1	1XStream		2JSsoup
2		2XStream	1JSoup
3		1JSoup	2XStream
4	2JSoup		1XStream
5		1XStream	2JSoup
6	2XStream		1JSoup
7	1JSoup		2XStream
8		2JSoup	1XStream

Na Tabela 1, a coluna 1 se refere ao identificador (id) do grupo e as colunas 2, 3 e 4 à ferramenta utilizada durante a sessão de depuração. O número que está antes do nome do programa corresponde à ordem de uso durante a depuração.

No Grupo 1 da Tabela 1 os estudantes utilizaram primeiro o programa XStream com a Jaguar Método e em seguida o programa JSoup com a CodeForest. No Grupo 2, os participantes utilizaram primeiro o JSoup com a CodeForest e em seguida do XStream com a Jaguar. Tal organização garante a aleatoriedade na distribuição dos participantes e mitiga a influência de fatores externos nos resultados do experimento como o cansaço, programa usado com cada ferramenta e a ordem de uso das ferramentas.

Antes do experimento foi ministrado um treinamento sobre o uso das ferramentas. Durante o experimento os estudantes tiveram 30 minutos para concluir cada tarefa, que consistiu na localização de um defeito em cada programa seguindo a distribuição da Tabela 1.

5.4 Questões do TAM

Elaboraram-se 10 questões para avaliar o potencial de uso das tecnologias com base no modelo TAM apresentado na Seção 2 deste artigo. As questões aplicadas sobre o uso de cada ferramenta são apresentadas na Tabela 2, bem como o grupo do TAM que cada questão pertence. Tanto para a CodeForest quanto para a Jaguar foi criado o mesmo grupo de questões.

Adotou-se a escala Likert como padrão de respostas para o questionário [6], considerando valores de 1 a 7, em que quando assinalado o valor 1 o participante concordou totalmente com a afirmação e 7 discordou totalmente da afirmação.

Tabela 2: Questões aplicadas

Questão	Grupo do TAM	Fator influenciado
Supondo que eu tive acesso à ferramenta, eu pretendo usá-la.	Intenção de uso	Intenção de uso futuro
Dado que eu tive acesso à ferramenta, eu imagino que a usaria.	Intenção de uso	Consideração de uso
Usar a ferramenta melhorou o meu desempenho durante a tarefa de depuração.	Utilidade percebida	Usabilidade
Usar a ferramenta durante a tarefa aumentou a minha produtividade.	Utilidade percebida	Produtividade
Usar a ferramenta melhorou a minha eficácia durante a atividade de depuração.	Utilidade percebida	Eficácia
De modo geral, eu achei a ferramenta útil para realizar a atividade de depuração.	Utilidade percebida	Utilidade da ferramenta
Aprender a ferramenta foi fácil para mim.	Facilidade de uso	Facilidade de aprendizado
Interagir com a ferramenta não exigiu muito do meu esforço mental.	Facilidade de uso	Esforço mental
De modo geral, eu achei a ferramenta fácil de usar.	Facilidade de uso	Facilidade de manuseio
Eu achei fácil contar com a ferramenta para realizar a tarefa de depuração.	Facilidade de uso	Facilidade da tarefa

6 RESULTADOS E DISCUSSÕES

6.1 Teste estatístico

Com o objetivo de responder à questão de pesquisa foram realizados testes de hipóteses para avaliar a percepção de usabilidade das ferramentas, medida por meio do modelo TAM para cada uma das questões e para a soma das respostas de cada participante para todas as questões avaliadas.

Buscou-se entender se os dados seguiam uma distribuição normal por meio do teste de *Shapiro-Wilk* [19] devido a sua eficiência para tamanhos de amostras se comparado aos resultados observados em outros testes. Ao avaliar os resultados do teste de *Shapiro-Wilk* e a representação dos dados por meio de histogramas verificou-se que eles não seguiam uma distribuição normal, portanto não seria adequada a utilização de uma abordagem paramétrica.

Sendo assim, foi utilizada uma abordagem não paramétrica por meio do teste de *Wilcoxon* pareado, em um intervalo de confiança de 95%. O teste avalia as diferenças entre as medianas das distribuições. Ou seja, ele avalia a diferença entre cada valor da amostra e o valor hipotético da mediana [12].

Na análise de dados foram considerados dois tratamentos. Embora a distribuição dos participantes apresentada na Tabela 1 possibilite três tratamentos, tanto a Jaguar método quanto a Jaguar

linha direcionam o desenvolvedor ao método e linha do defeito. Ou seja, elas representam apenas versões da Jaguar e não mudam a quantidade de tratamentos aplicáveis no estudo.

Primeiramente buscou-se avaliar se há diferenças entre as ferramentas. As hipóteses consideradas são listadas a seguir. A Tabela 3 apresenta os resultados obtidos.
H_0: há igualdade para a percepção de usabilidade entre as ferramentas CodeForest e Jaguar.
H_1: há diferenças para a percepção de usabilidade entre as ferramentas CodeForest e Jaguar.

Tabela 3: Teste de *Wilcoxon* para a igualdade

Questão	Resultado (valor-p)
Supondo que eu tive acesso a ferramenta, eu pretendo usá-la.	0,06721
Dado que eu tive acesso a ferramenta, eu imagino que a usaria.	0,01173
Usar a ferramenta melhorou o meu desempenho durante a tarefa de depuração.	0,01006
Usar a ferramenta durante a tarefa aumentou a minha produtividade.	0,02485
Usar a ferramenta melhorou a minha eficácia durante a atividade de depuração.	0,06208
De modo geral, eu achei a ferramenta útil para realizar a atividade de depuração.	0,04758
Aprender a ferramenta foi fácil para mim.	0,03745
Interagir com a ferramenta não exigiu muito do meu esforço mental.	0,07724
De modo geral, eu achei a ferramenta fácil de usar.	0,07659
Eu achei fácil contar com a ferramenta para realizar a tarefa de depuração.	0,04537
Resultados completos considerando a soma das respostas do TAM.	0,02054

Verificaram-se diferenças estatísticas em 6 das 10 questões do TAM. Para os resultados completos (com todas as questões), também foram observadas diferenças estatísticas. Em outras palavras, considerando que a hipótese nula foi rejeitada avaliou-se qual ferramenta obteve maior usabilidade. Adotaram-se as hipóteses descritas a seguir. A Tabela 4 descreve os resultados obtidos.
H_0: A CodeForest obteve maior usabilidade do que a Jaguar.
H_1: A Jaguar obteve maior usabilidade do que a CodeForest.

Em todas as questões, rejeitou-se a hipótese nula e aceitou-se a hipótese alternativa de que a Jaguar apresentou maior usabilidade do que a CodeForest. Quando avaliados os resultados completos do TAM a Jaguar também foi superior. Ou seja, a Jaguar obteve maior usabilidade do que a CodeForest.

6.2 Tamanho do efeito

Além do teste estatístico é importante considerar o tamanho do efeito que os resultados obtidos representam sobre a amostra. O tamanho do efeito, em inglês *Effect Size*, avalia a magnitude da diferença entre os grupos [20]. O nível de significância estatística

Tabela 4: Avaliação das diferenças entre as ferramentas

Questão	Resultado (valor-p)
Dado que eu tive acesso a ferramenta, eu imagino que a usaria.	0,005863
Usar a ferramenta melhorou o meu desempenho durante a tarefa de depuração.	0,00503
Usar a ferramenta durante a tarefa aumentou a minha produtividade.	0,01243
De modo geral, eu achei a ferramenta útil para realizar a atividade de depuração.	0,02379
Aprender a ferramenta foi fácil para mim.	0,01873
Eu achei fácil contar com a ferramenta para realizar a tarefa de depuração.	0,02268
Resultados completos considerando a soma das respostas do TAM.	0,01027

é afetado por diversas características; entretanto, o tamanho da amostra é a mais determinante [20].

Tendo em vista esta necessidade, optou-se por utilizar a medida *Delta de Cliff* para avaliar o tamanho do efeito nas questões em que rejeitou-se a hipótese de igualdade entre as ferramentas [4].

A Tabela 5 apresenta com base nas ideias propostas por Romano [18] e Cohen [5] a escala utilizada para a avaliação do tamanho do efeito. A Tabela 6 apresenta os resultados obtidos para cada uma das questões.

Tabela 5: Escala de avaliação do tamanho do efeito

Valor	Interpretação
Inferior a 0,147	Diferença negligenciavel, ou seja, não há diferença entre os grupos
Entre 0,33 a 0,474	Diferença média entre os grupos
Maior ou igual a 0,474	Diferença grande entre os grupos

Tabela 6: Tamanho do efeito sobre a amostra

Questão	Valor observado
Dado que eu tive acesso a ferramenta, eu imagino que a usaria.	0,21504
Usar a ferramenta melhorou o meu desempenho durante a tarefa de depuração.	0,18865
Usar a ferramenta durante a tarefa aumentou a minha produtividade.	0,14967
De modo geral, eu achei a ferramenta útil para realizar a atividade de depuração.	0,1324
Aprender a ferramenta foi fácil para mim.	0,16514
Eu achei fácil contar com a ferramenta para realizar a tarefa de depuração.	0,13498
Resultados completos considerando a soma das respostas do TAM.	0,17515

É possível perceber que a representatividade da diferença sobre a amostra pode ser considerada pequena em quatro das seis afirmações e para a soma das respostas dos participantes. Nas outras duas afirmações pode ser considerada insignificante. Em outras palavras, há diferença estatística para a percepção de usabilidade entre as ferramentas CodeForest e Jaguar, em que a Jaguar foi melhor avaliada em relação a CodeForest; entretanto ao avaliar a representatividade dessa diferença sobre a amostra a diferença pode ser considerada pequena entre as ferramentas.

Diferentemente dos resultados parciais, em que não foram identificadas diferenças entre a usabilidade das ferramentas por meio de testes estatísticos [7], os dados completos indicam que a Jaguar obteve maior usabilidade. É importante reforçar que grande parte das pessoas que participaram do primeiro experimento ainda estavam na metade do curso, podendo acarretar em diferenças nos resultados.

Comparando-se os valores observados no teste de Wilcoxon, no primeiro experimento não foram observadas diferenças estatísticas para nenhuma das questões [7]. Enquanto nos resultados completos foram observadas diferenças estatísticas em 6 das 10 questões avaliadas. Os resultados completos permitiram a avaliação da representatividade da diferença sobre a amostra, considerada pequena; mas com significância estatística importante para os resultados.

7 AMEAÇAS À VALIDADE

Os resultados de um projeto de pesquisa são influenciados por fatores que afetam a validade externa, interna e de construto.

A validade externa indica a capacidade dos resultados serem generalizados para outros trabalhos. Os resultados sobre o uso de metáforas visuais e textuais de depuração limitam-se às ferramentas CodeForest e Jaguar. Existem outras ferramentas de depuração, como a Tarantula [11] e a GZoltar [9] que não foram consideradas devido à necessidade de um trabalho conjunto com pesquisadores de outros países para o planejamento do experimento.

A validade interna avalia as condições básicas aplicáveis ao estudo. A condução do estudo ocorreu apenas com alunos de curso de graduação que não possuem experiência profissional significativa. O grau de dificuldade dos defeitos foi médio para ambas as ferramentas; entretanto, é complexo mensurar a influência dessa dificuldade nos resultados. É importante considerar que os resultados foram descritos apenas em relação ao modelo TAM.

A validade de construto verifica o quanto uma medida está correlacionada com o grupo de variáveis de interesse. À percepção de usabilidade das ferramentas CodeForest e Jaguar foi analisada por meio de uma abordagem não paramétrica com o teste pareado de *Wilcoxon* e com o Delta de Cliff para mensurar a representatividade da diferença sobre a amostra.

8 CONCLUSÕES

Este artigo apresentou os resultados de um estudo realizado com as ferramentas CodeForest e Jaguar para avaliar a percepção de usabilidade das ferramentas para a localização de defeitos, em uma avaliação experimental com 119 estudantes baseada no modelo TAM.

A análise dos resultados foi realizada por meio de uma abordagem não paramétrica com o teste pareado de *Wilcoxon*. Posteriormente verificou-se o tamanho do efeito que a diferença identificada entre as ferramentas representa sobre a amostra de pessoas que participaram do estudo.

Os resultados demonstraram que há indícios de que a Jaguar possuí maior usabilidade do que a CodeForest para a localização de defeitos. Entretanto a diferença é pequena dado o tamanho da amostra. Ou seja, há diferença entre as ferramentas; mas a representatividade dado o conjunto de pessoas que participaram do estudo é pequena.

Os resultados observados neste estudo podem ser usados como referência para o desenvolvimento e avaliação de novas ferramentas visuais de depuração, bem como auxiliar no processo de transformação digital das organizações com a adoção de novas tecnologias. Neste cenário considera-se como principais fatores a ausência de trabalhos na literatura com este objetivo, as dificuldades enfrentadas pelos desenvolvedores na realização da atividade de depuração e os altos custos para as organizações com correção de defeitos no processo evolutivo de um sistema.

REFERÊNCIAS

[1] Rui Abreu, Peter Zoeteweij, and Arjan J. C. van Gemund. 2007. On the accuracy of spectrum-based fault localization. In *Proceedings of the Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION*. 89–98.
[2] Keijiro Araki, Zengo Furukawa, and Jun Cheng. 1991. A General Framework for Debugging. *IEEE Software Magazine* v.8, 3 (1991).
[3] Ronald S. Calinger. 2015. *Leonhard Euler: Mathematical Genius in the Enlightenment.* Princeton University Press, Princeton, NJ, USA.
[4] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological Bulletin* 114, 3 (Nov. 1993), 494–509. http://search.ebscohost.com/login.aspx?direct=true&db=pdh&AN=bul-114-3-494&site=ehost-live
[5] J. Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences.* Lawrence Erlbaum Associates.
[6] R. A. Cummins and E. Gullone. 2000. Why we should not use 5-point Likert scales: The case for subjective quality of life measurement.. In *Proceedings, Second International Conference on Quality of Life in Cities* (Singapore: National University of Singapore.). 74–93.
[7] Fabio Pereira da Silva, Higor Amario de Souza, and Marcos Lordello Chaim. 2018. Avaliação de usabilidade de ferramentas de depuração de software. In *Anais do XIV Simpósio Brasileiro de Sistemas de Informação* (Caxias do Sul). SBC, Porto Alegre, RS, Brasil, 495–489. https://sol.sbc.org.br/index.php/sbsi/article/view/5124
[8] Fred D. Davis. 1989. Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Q.* 13, 3 (sep 1989), 319–340.
[9] Carlos Gouveia, José Campos, and Rui Abreu. 2013. Using HTML5 visualizations in software fault localization. In *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*. 1–10. https://doi.org/10.1109/VISSOFT.2013.6650539
[10] J. A. Jones, M. J. Harrold, and J. Stasko. 2002. Visualization of test information to assist fault localization. In *Proceedings of the 24th International Conference on Software Engineering*. 467–477.
[11] James A. Jones, Mary Jean Harrold, and John Stasko. 2002. Visualization of Test Information to Assist Fault Localization. *Proceedings of the 24th ACM/IEEE International Conference on Software Engineering* (2002). http://dl.acm.org/citation.cfm?id=1273468
[12] Leonardo J. Kazmier. 2007. *Estatística aplicada a administração e economia* (4a ed.). Bookman, Porto Alegre.
[13] Paul Legris, John Ingham, and Pierre Collerette. 2003. Why do people use information technology? A critical review of the technology acceptance model. *Information & Management* 40, 3 (2003), 191–204.
[14] Danilo Mutti. 2014. *Coverage Based Debugging Visualization.* Master's thesis. Universidade de São Paulo.
[15] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller. 2017. Evaluating and Improving Fault Localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 609–620. https://doi.org/10.1109/ICSE.2017.62
[16] Manos Renieris and Steven P. Reiss. 2003. Fault Localization With Nearest Neighbor Queries.. In *ASE*. IEEE Computer Society, 30–39.
[17] Henrique Lemos Ribeiro. 2016. *On the use of control- and data-flow in fault localization.* Master's thesis. Universidade de São Paulo.
[18] J. Romano, J.D. Kromrey, J. Coraggio, and J. Skowronek. 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for

evaluating group differences on the NSSE and other surveys?. In *annual meeting of the Florida Association of Institutional Research*. 1–3.

[19] S. S. Shapiro and M. B. Wilk. 1965. An Analysis of Variance Test for Normality (Complete Samples). *Biometrika* 52, 3/4 (Dec. 1965), 591–611. http://links.jstor.org/sici?sici=0006-3444%28196512%2952%3A3%2F4%3C591%3AAAOVTF%3E2.0.CO%3B2-B

[20] Gail Sullivan and Richard Feinn. 2012. Using Effect Size—or Why the P Value Is Not Enough. *Journal of Graduate Medical Education* 4, 3 (2012), 279–282. https://doi.org/10.4300/JGME-D-12-00156.1 arXiv:https://doi.org/10.4300/JGME-D-12-00156.1

[21] L. Taylor, R. Titmuss, and C. Lebre. 1999. The challenges of seamless handover in future mobile multimedia networks. *IEEE Personal Communications* 6, 2 (Apr 1999), 32–37.

[22] Mark Weiser. 1981. Program Slicing. In *Proceedings of the 5th International Conference on Software Engineering* (San Diego, California, USA) *(ICSE '81)*. IEEE Press, 439–449.

[23] Andreas Zeller. 2005. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers Inc.