Debugging visualization tools: A systematic review

Fabio Pereira da Silva[1], Higor Amario de Souza[1], Marcos Lordello Chaim[1], and Edson Saraiva de Almeida[2]

[1] University of São Paulo
[2] São Paulo State Faculty of Technology East Zone

Abstract Debugging is the task of locate and fix program bugs. Debugging activity is performed in the same way since 1960's, when the first symbolic debuggers were introduced. Recently, visualization techniques have been proposed to represent program information during fault localization. However, none of them were introduced at industrial environments. This article presents a systematic review about visualization techniques for debugging. Despite the increasing number of studies in the area, visual debugging tools are not yet used in practice.

Keywords: Debugging Tools · Software Visualization · User Experience

1 Introdução

A Engenharia de Software é uma área relacionada com todos os aspectos de produção de software [21]. Ela evoluiu significativamente nas últimas décadas procurando estabelecer técnicas, critérios, métodos e ferramentas para a produção de aplicações, resultante do aumento significativo da utilização de sistemas computacionais em praticamente todas as áreas, o que provoca uma crescente demanda por qualidade e produtividade [21]. Uma das principais técnicas utilizadas é a depuração responsável pela localização e correção de defeitos gerados durante o desenvolvimento de programas [14].

Estima-se que 50 a 70% do esforço de desenvolvimento de software é despendido após a sua entrega ao cliente, ou seja, na fase de manutenção [14]. A depuração em geral é realizada por meio de depuradores simbólicos que visam acompanhar a execução do programa [14].

A depuração é uma consequência do teste bem sucedido, isto é, quando um caso de teste apresenta uma falha [21]. Esse teste é avaliado até que uma divergência entre o resultado esperado e os valores reais obtidos seja encontrada [14]. Na maioria das vezes os resultados esperados representam um sintoma interno de uma causa subjacente ou defeito; porém, desconhecida [14]. Sintomas internos são caracterizados como os valores de uma ou mais variáveis em determinado ponto de execução do programa [3].

Diante do aumento significativo da complexidade dos sistemas que estão sendo desenvolvidos ou da incorporação constante de novas funcionalidades nas aplicações já existentes ferramentas que auxiliem na localização de defeitos são de grande importância [6].

Entretanto, não houve mudanças significativas na forma como ela é realizada ao longo das últimas décadas e, na prática, ainda é executada manualmente [6]. Diante deste problema, novas ferramentas estão sendo propostas com o objetivo de auxiliar os desenvolvedores na localização de defeitos [6]. Sendo assim, o uso de ferramentas visuais de depuração pode colaborar para que os defeitos sejam solucionados em menor período de tempo [11].

Dentro desse contexto torna-se necessário o desenvolvimento de novas ferramentas que possam auxiliar na atividade de depuração, visando diminuir o tempo gasto para a localização de defeitos [6]. Porém, grande parte das ferramentas propostas acabam não fazendo experimentos com profissionais e as que possuem não realizaram avaliações sobre a usabilidade, eficácia e eficiência dos programadores durante o uso de ferramentas visuais [16]. Estes problemas mencionados acabam dificultando que projetos de ferramentas que auxiliem os desenvolvedores durante a atividade de depuração sejam levados a indústria de software [9, 16].

O objetivo deste trabalho é identificar estudos relacionados ao desenvolvimento de novas ferramentas visuais que possam auxiliar os desenvolvedores durante a depuração. Este trabalho oferece como principais contribuições uma visão abrangente do estado da arte inerente ao desenvolvimento de ferramentas visuais que auxiliem na localização de defeitos.

O restante deste artigo está organizado da seguinte forma: na Seção 2 são descritos os materiais e métodos utilizados. A Seção 3 apresenta os resultados obtidos. A Seção 4 dedica-se às discussões dos resultados obtidos. Na Seção 5 são detalhadas as ameaças à validade deste estudo. Ao final, na Seção 6 são apresentadas as considerações finais, bem como suas contribuições e limitações.

2 Materiais e métodos

As revisões sistemáticas são desenhadas para serem metódicas, explícitas e passíveis de reprodução [10]. Visam auxiliar na identificação de rumos para futuros estudos através da análise do estado da arte de uma dada área da ciência [10].

2.1 Critérios de seleção de trabalhos

Este trabalho tem como objetivos a identificação das evoluções ocorridas nos últimos anos no desenvolvimento de ferramentas de visualização que auxiliem os desenvolvedores na atividade de depuração. Avaliamos os métodos, técnicas adotadas e problemas enfrentados com as aplicações propostas para a sua utilização em ambientes reais de desenvolvimento de software.

Para direcionar a pesquisa foram estabelecidas algumas questões a serem respondidas ao término desta revisão:

1. Quais foram as ferramentas visuais ou gráficas propostas ao longo dos últimos anos com o objetivo de auxiliar os desenvolvedores na atividade de depuração?

2. Alguma das ferramentas apresentadas teve a sua usabilidade, eficácia e eficiência avaliada em ambientes reais?

3. Foram encontrados trabalhos que utilizam técnicas de Interação Humano Computador para a construção de ferramentas visuais alinhadas as expectativas dos seus usuários?

4. O que ainda falta para que ferramentas visuais de depuração sejam adotadas na indústria de software?

Para a realização da busca de trabalhos relacionados foram consideradas as seguintes palavras chaves: "depuração", relacionada com os termos *debugging*, *fault localization* e *coverage*; "visualização", associada aos termos *software visualization*, *visualization of debugging information* e *debugging tool*; "avaliação", associada aos termos *evaluation* e *assessment*; e "experiência de usuário", associada com o termo *user experience*. Foram adotados os critérios de inclusão e exclusão descritos na Tabela 1.

Tabela 1. Critérios de Inclusão/Exclusão

Critérios de inclusão	Critérios de exclusão
Trabalhos disponíveis integralmente em bases de dados científicas ou em versões impressas.	Trabalhos que não avaliem as técnicas de depuração ou que não apresentem novas abordagens para esta atividade.
Trabalhos que tragam propostas de melhorias à atividade de depuração, seja pelo desenvolvimento de novas ferramentas, seja por estudos com desenvolvedores durante a sua utilização.	Trabalhos que não apresentem testes estatísticos que comprovem a eficiência do método aplicado.
Trabalhos que apresentem comparações entre depuradores simbólicos com ferramentas visuais.	Trabalhos que não utilizem técnicas de visualização para a atividade de depuração.
Trabalhos que realizem comparações de ferramentas visuais que auxiliem os desenvolvedores na atividade de depuração.	Trabalhos que não sejam aderentes ao tema proposto.

Os trabalhos que atenderam aos critérios de inclusão foram categorizados em projetos que envolvam os seguintes tópicos:

- Trabalhos que abordaram o desenvolvimento de ferramentas de visualização que auxiliem os desenvolvedores durante a depuração.
- Trabalhos que realizaram análises estatísticas detalhadas que comprovem a eficiência do método aplicado.
- Trabalhos que apresentaram o uso de técnicas de Interação Humano Computador para entendimento de como as pessoas executam a atividade de depuração.
- Trabalhos com ênfase na construção de ferramentas de reengenharia que auxiliem na localização de defeitos.

– Trabalhos que abordem o uso de ferramentas para auxiliar na depuração de sistemas distribuídos.

2.2 Condução da revisão da sistemática

Seguindo os critérios de seleção, a revisão foi conduzida por um período de três meses (Abril/2015 a Agosto/2015) e refeita em Dezembro de 2018 para identificar se houve algum trabalho relevante desenvolvido nesse período. Para a obtenção dos estudos primários foi formada a seguinte *string* de busca aplicada no repositório digital da IEEE com base nas palavras chaves: *(debugging) or ((software visualization) and (visualization of debugging information) and (debugging tool)) or ((fault localization) and (coverage)) or (user experience).*

O repositório digital da IEEE foi escolhido por ser uma base de referência para estudos na área de Engenharia de Software.

Ao todo foram localizados 99 trabalhos e 20 deles foram incluídos após a leitura do título e do resumo dos artigos, considerando os critérios de inclusão e exclusão. Diante do baixo número de ferramentas visuais de depuração encontradas foram incluídos também trabalhos que apresentem alguma forma gráfica de exibição de informações durante a depuração.

A Tabela 2 apresenta os trabalhos incluídos. Nas próximas seções serão descritas visões detalhadas dos trabalhos, como eles foram distribuídos e a relevância para os resultados desta revisão.

3 Resultados

Esta seção dedica-se à discussão dos resultados obtidos. Ela é baseada nas contribuições das informações para a atividade de depuração.

3.1 Principais ferramentas encontradas

GZoltar A GZoltar é uma ferramenta visual com o objetivo de auxiliar os desenvolvedores na atividade de depuração por meio de representações bidimensionais construídas em HTML5 que guiam o programador até a localização do defeito [6].

A aplicação é um *plugin* integrável com o IDE Eclipse[3] voltada para programas desenvolvidos na linguagem Java e pode ser utilizada simultaneamente com casos de testes automatizados com o *framework* JUnit[4] [16].

A GZoltar necessita das informações de cobertura dos casos de teste para realizar a localização de defeitos utilizando a heurística Ochiai [6]. 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 [6].

[3] http://www.eclipse.org
[4] https://www.junit.org

Tabela 2. Artigos incluídos

Título	Ano
A debugging and testing tool for supporting software evolution [1].	1995
A lightweight awareness service for industrial environments [13].	1997
XSUDS-SDL: a tool for diagnosis and understanding software specifications [12].	1999
Language-agnostic program rendering for presentation, debugging and visualization [4].	2000
Gemini: maintenance support environment based on code clone analysis [23].	2002
Visualization of test information to assist fault localization [9].	2002
Case study: Visual debugging of finite element codes [5].	2002
Visualizing multiple program executions to assist behavior verification [25].	2009
How programmers debug, revisited: An information foraging theory perspective [11].	2010
Guided test visualization: Making sense of errors in concurrent programs [24].	2011
Bug Maps: A tool for the visual exploration and analysis of bugs [8].	2012
Model-Level Debugging of embedded real-time systems [7].	2012
A visual studio plug-in for CProver [20].	2013
Monitoring user interactions for supporting failure reproduction [18].	2013
A perspective on the evolution of live programming [22].	2013
Development and debugging of distributed virtual reality applications [15].	2013
Cues for scent intensification in debugging [16].	2013
Visual programming of MPI applications: Debugging and performance analysis [2].	2013
The challenge of helping the programmer during debugging [17].	2014
In *Bug: Visual analytics of bug repositories [19].	2014

Os dados são coletados por meio de heurísticas, também chamadas de métricas. As métricas utilizam as informações de cobertura para inferir os elementos de um programa mais suspeitos de conterem defeitos [9]. Elas são baseadas em coeficientes que levam em consideração os componentes que foram, ou não foram, executados pelos casos de testes.

A GZoltar utiliza a técnica de *Localização de defeitos baseada em cobertura de código* Para Renieris e Reiss [6], a técnica, é definida por um conjunto de componentes (comandos, blocos, predicados, associações definição-uso ou subprogramas) cobertos durante a execução de um teste.

Primeiramente a GZoltar executa os casos de testes, coletando as informações do programa a partir da aplicação da heurística Ochiai. Em seguida a ferramenta classifica os elementos com maior grau de risco de apresentar o defeito representando-os graficamente.

A GZoltar possui um esquema de cores que indica a probabilidade de cada elemento do programa conter defeitos. As informações da aplicação podem ser

visualizadas de três maneiras: representação em anel, particionamento vertical e ordenação hierárquica. 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 [6].

A ferramenta oferece uma visão abrangente e integrada de um projeto de software com um esquema de navegação intuitivo. Com base nestas informações, os desenvolvedores podem inspecionar os trechos do programa mais suspeitos de conter defeitos [6].

Dos trabalhos encontrados na literatura, a GZoltar foi a única ferramenta avaliada em relação a sua eficácia e eficiência para a localização de defeitos. Durante o experimento todos os desenvolvedores conseguiram realizar a localização do defeito com o uso da ferramenta. Quando a busca do defeito ocorreu apenas com o uso de depuradores simbólicos somente 35% deles conseguiram encontrá-lo [6].

O experimento foi realizado com quarenta pessoas, reforçando a necessidade do desenvolvimento de novos trabalhos que avaliem ferramentas visuais para depuração com profissionais da indústria. Nas Figuras 1 e 2, são apresentadas as formas de exibições que podem ser utilizadas com a ferramenta.

Figura 1. Representação em anel e particionamento vertical da GZoltar

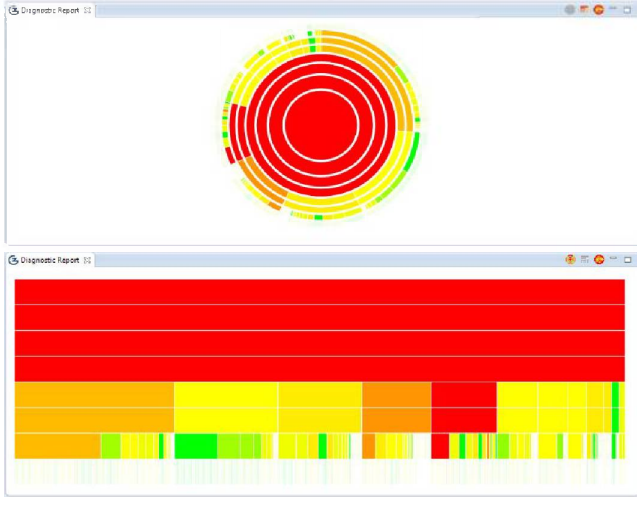
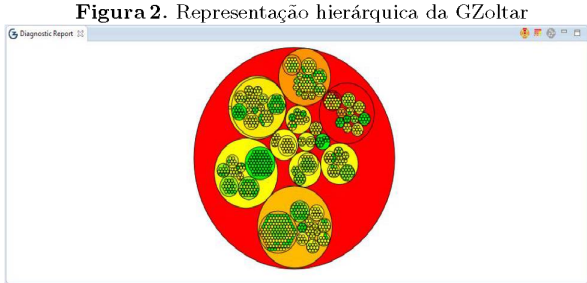


Figura 2. Representação hierárquica da GZoltar



Tarantula Jones et al. [9] 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 os elementos mais suspeitos de um programa.

A 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. As Figuras 3 e 4 apresentam a ferramenta.

Figura 3. Representação da ferramenta Tarantula

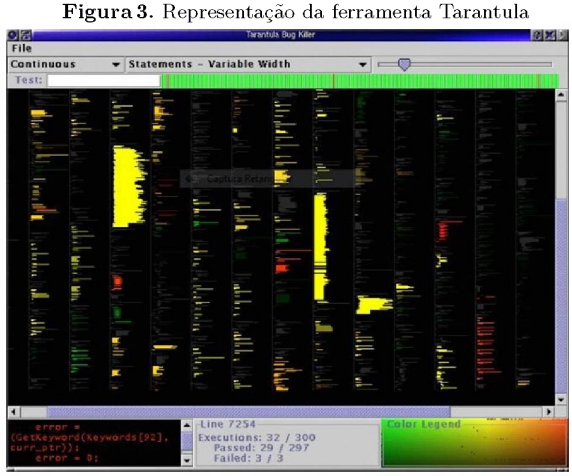
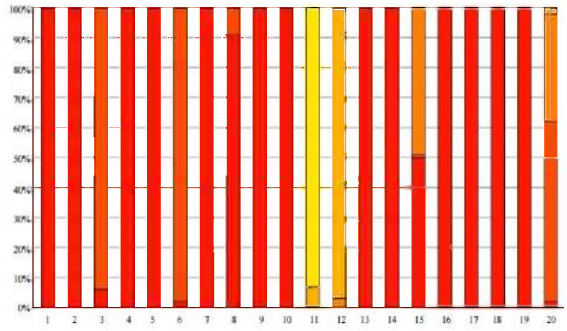


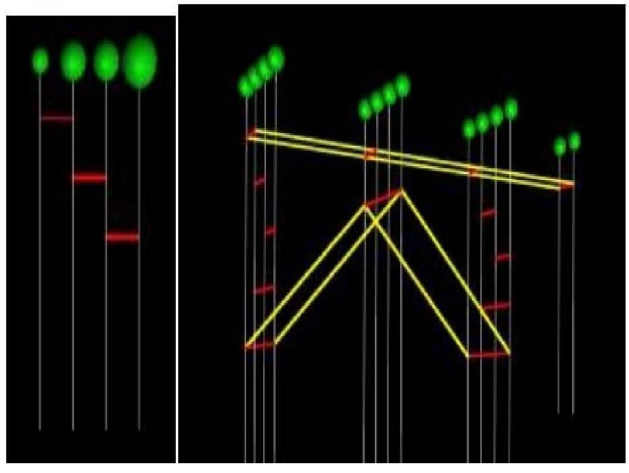
Figura 4. Esquema de coloração utilizado pela ferramenta



Visualização em múltiplos planos 3D A ferramenta proposta por Zhao et al. [25] tem como objetivo auxiliar os desenvolvedores na atividade de depuração combinando múltiplos planos tridimensionais.

A aplicação permite comparações entre múltiplas execuções de um programa, auxiliando os desenvolvedores para que haja uma melhor compreensão do código fonte. A ferramenta compara o tempo de execução de um programa com diferentes entradas. Isto permite um melhor projeto dos testes, análise do tempo de resposta da aplicação, identificação de duplicidades nos casos de testes e auxílio na depuração [25].

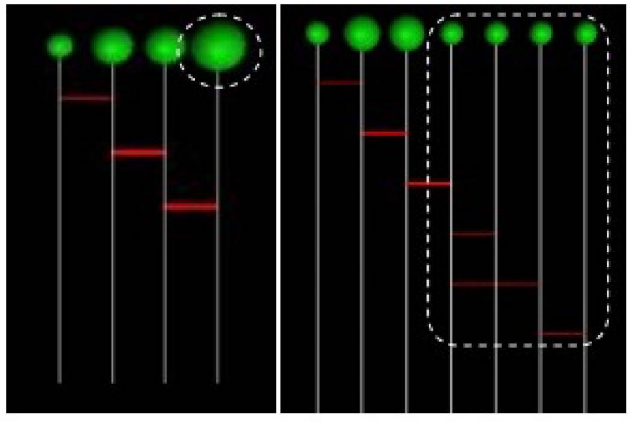
Figura 5. Representação gráfica através de diagramas de sequência



Foram utilizados diagramas de sequência para realizar as demonstrações das execuções em 2D. Cada objeto é representado por uma pequena esfera sólida, podendo estar relacionado a outros objetos dependendo do nível de abstração utilizado. Chamadas de métodos foram representadas por bordas dispostas de acordo com a execução dos eventos. O mapeamento entre as execuções foi realçado com linhas amarelas [25]. A Figura 5 demonstra um diagrama de sequência gerado pela aplicação.

Na exibição em 3D, são utilizados alvos visuais para capturar os pontos focais dos programadores, conforme pode ser visto na Figura 6 nos trechos tracejados em branco. O tamanho da esfera é diferente visando manter o foco do programador durante a depuração nos elementos mais suspeitos do programa.

Figura 6. Análise dos pontos do diagrama focados pelo usuário

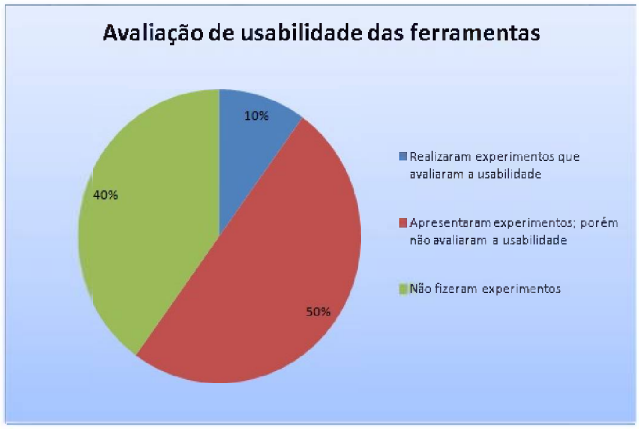


3.2 Avaliações de usabilidade

Neste estudo, somente os trabalhos de Jones et al. [9] e Gouveia [6] apresentaram resultados de avaliações com desenvolvedores. Enquanto os demais focaram somente nas características da aplicação sem avaliações de sua usabilidade, eficácia e eficiência.

Tal característica sustenta o que foi mencionado na introdução deste trabalho, que além da ausência de propostas que abordem o uso de ferramentas visuais durante a atividade de depuração, faltam experimentos com usuários que avaliem as ferramentas abordadas em situações reais de desenvolvimento. Outro fator importante verificado é a ausência de estudos que verifiquem o quanto elas realmente podem colaborar para a realização da atividade de depuração pelos desenvolvedores. A Figura 7 demonstra os resultados obtidos.

Figura 7. Avaliação da usabilidade das ferramentas

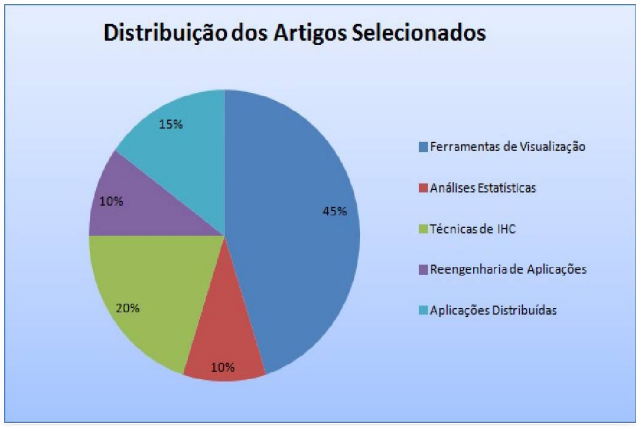


3.3 Avaliação das técnicas empregadas nos artigos

Os artigos foram categorizados em trabalhos que utilizaram ferramentas de visualização, análises estatísticas, avaliações com desenvolvedores por meio de técnicas de Interação Humano Computador, reengenharia de aplicações e ferramentas para auxiliar a depuração em sistemas distribuídos.

Algumas ferramentas de visualização possuem ênfase em mais de uma categoria avaliada nesta revisão. Por isso, foram categorizados no item correspondente às categorias adicionais, e não em ferramentas de visualização. A Figura 8 detalha como foram distribuídos os artigos selecionados.

Figura 8. Distribuição dos artigos selecionados



3.4 Análise do período dos artigos selecionados

Foram considerados artigos desenvolvidos a partir do ano de 1990. Foi possível verificar que houve um aumento significativo de pesquisas sobre a atividade de depuração nos últimos anos. Dos vinte artigos selecionados, 60% deles foram publicados entre 2010 e 2018, 25% entre 2000 e 2009 e 15% entre 1990 e 1999, o que demonstra o crescimento recente de pesquisas sobre o desenvolvimento de ferramentas visuais.

4 Discussão

Nesta seção, serão discutidos os resultados obtidos em relação a cada uma das questões de pesquisa descritas na Seção 2.1.

4.1 Ferramentas visuais propostas

Ao longo dos últimos anos foram propostas algumas aplicações com o objetivo de auxiliar os desenvolvedores na atividade de depuração, em especial com o desenvolvimento de ferramentas como a Tarantula [9], GZoltar [6], Code Bubbles [17], Bug Maps [8] e In *Bug [19].

Foram localizadas duas ferramentas que exibem os elementos mais suspeitos através de representações bidimensionais nos trabalhos de Jones et al. [9] e Gouveia [6].

4.2 Avaliação em ambientes reais

Somente os artigos de Jones et al. [9] e Gouveia [6] conduziram experimentos com ferramentas de visualização para avaliar a usabilidade, eficiência e eficácia das aplicações.

A ferramenta Tarantula [9], foi avaliada quanto a sua capacidade de colorir os elementos que possuem maior probabilidade de conter defeitos. Embora tenham sido observados resultados promissores, o experimento foi bastante restrito a avaliação da metáfora visual proposta sem considerar outras ferramentas.

A avaliação da ferramenta GZoltar [6], buscou realizar uma comparação sobre o uso de ferramentas visuais em relação aos depuradores simbólicos. Dentre os participantes que utilizaram a ferramenta, todos eles conseguiram realizar a localização do defeito no tempo previsto. Considerando os desenvolvedores que não utilizaram a GZoltar, a grande maioria não chegou nem mesmo próximo ao defeito e apenas 35% dos desenvolvedores conseguiram localizá-lo [16].

Entretanto, a avaliação também foi bastante restrita tanto em número de defeitos avaliados (apenas um) e na comparação com ferramentas textuais, tornando necessário o desenvolvimento de novos estudos.

4.3 Uso de técnicas de Interação Humano Computador

O trabalho de Lawrance, et al. [11] avalia como os desenvolvedores realizam a atividade de depuração usando modelo Caça-Caçador.

No estudo, foi identificado primeiramente como os desenvolvedores avaliam as hipóteses encontradas durante a depuração. Vídeos foram gravados com os participantes para entender como as suas ações e expressões faciais estão relacionadas com o surgimento de novas hipóteses. O estudo serve de subsídio para a identificação de fatores comportamentais, observados nos programadores, úteis para o *design* de novas ferramentas.

O artigo de Tanimoto [22], tem como objetivo o uso da técnica de *Liveness* na construção de novas ferramentas visuais. A técnica proposta serve como subsídio para o entendimento de como os programadores constroem o seu modelo mental durante a depuração.

Considerando os estudos dos artigos de Jones et al. [9] e Gouveia [6], somente 20% de todos os trabalhos levaram em consideração a avaliação das experiências dos usuários durante a construção e a avaliação de ferramenta visuais ou apresentaram propostas de avaliação da eficácia e eficiência de ferramentas visuais.

4.4 Dificuldades para introdução na indústria de software

O aumento significativo de pesquisas sobre a atividade depuração nos últimos anos indica que se trata de um tema relevante. Porém, poucas delas são validadas com desenvolvedores. A razão, em parte, está na dificuldade de uso em ambientes industriais, seja por possuírem um baixo grau de usabilidade, seja a menor eficácia em relação aos depuradores simbólicos, seja pelo grande consumo de tempo em sua utilização.

5 Ameaças à validade

As ameças à validade de um projeto de pesquisa podem ser ameaças à validade interna, externa e de construto.

A validade interna avalia as condições básicas aplicáveis ao estudo. Neste trabalho foram localizados poucos trabalhos que introduziram o uso de ferramentas visuais durante a atividade de depuração. É importante considerar que foi utilizado apenas o repositório digital da IEEE. Outros trabalhos poderiam ter sido encontrados caso tivessem sido avaliadas bases como ACM, Scopus, Research Gate e Springer. Entretanto, os artigos encontrados já relatam a ausência de ferramentas visuais para depuração existentes na literatura.

Como ameaça à validade externa está a capacidade de generalização dos resultados. Não é possível generalizá-los para todas as ferramentas de depuração, pois o foco do trabalho centralizou-se no estudo de ferramentas visuais.

6 Conclusões

Este trabalho apresentou uma revisão sistemática para entender as evoluções ao longo dos últimos anos no desenvolvimento de ferramentas visuais para depuração, os principais desafios a serem enfrentados e como as novas ferramentas desenvolvidas podem contribuir para a execução desta atividade.

A leitura das obras possibilitou a identificação de trabalhos relevantes para o objetivo proposto nesta revisão, permitindo que as questões de pesquisa fossem respondidas e avaliar o estado da arte sobre o desenvolvimento de ferramentas visuais para depuração.

A partir da revisão sistemática é oferecida como principal contribuição a constatação de que apesar do aumento considerável de estudos desenvolvidos nos últimos anos, grande parte das ferramentas que visam auxiliar os programadores na localização de defeitos acabam nunca sendo utilizadas em ambientes reais. Seja por deixarem de lado avaliações com desenvolvedores ou por não realizarem estudos com outras técnicas de exibições de informações.

O trabalho pode ser utilizado por pesquisadores e engenheiros de software que visem obter maior entendimento inerente as principais dificuldades enfrentadas e evoluções ao longo dos últimos anos na atividade de depuração ou interessados na construção de novas ferramentas visuais.

Referências

1. Abramson, D., Sosic, R.: A debugging and testing tool for supporting software evolution. Automated Software Engineering **3**(3), 369–390 (Aug 1996). https://doi.org/10.1007/BF00132573
2. Böhm, S., Běhálek, M., Meca, O., Šurkovský, M.: Visual programming of mpi applications: Debugging and performance analysis. In: 2013 Federated Conference on Computer Science and Information Systems. pp. 1495–1502 (Sept 2013)
3. Chaim, M.L., Maldonado, J.C., Jino, M.: Processo de depuração depois do teste: Definição e análise (2002)
4. Collberg, C.S., Davey, S., Proebsting, T.A.: Language-agnostic program rendering for presentation, debugging and visualization. In: Proceeding 2000 IEEE International Symposium on Visual Languages. pp. 183–190 (2000). https://doi.org/10.1109/VL.2000.874382
5. Crossno, P., Rogers, D.H., Garasi, C.J.: Case study: Visual debugging of finite element codes. IEEE Visualization VIS - 2002 (2002)
6. Golveia, C., Campos, J., Abreu, R.: Using html5 visualizations in software fault localization. 1st IEEE Working Conference on Software Visualization (2013)
7. Haberl, W., Herrmannsdoerfer, M., Birke, J., Baumgarten, U.: Model-level debugging of embedded real-time systems. In: 2010 10th IEEE International Conference on Computer and Information Technology. pp. 1887–1894 (June 2010)
8. Hora, A., Anquetil, N., Couto, C., Valente, M.T., Martins, J.: Bugmaps: A tool for the visual exploration and analysis of bugs. 16th European Conference on Software Maintenance and Reengineering (CSMR) (2012)
9. Jones, J.A., Harrold, M.J., Stasko, J.: Visualization of test information to assist fault localization. Proceedings of the 24th ACM/IEEE International Conference on Software Engineering (2002)

10. Kitchenham, B.A., Budgen, D., Brereton, P.: Evidence-Based Software Engineering and Systematic Reviews. Chapman & Hall/CRC (2015)
11. Lawrance, J., Bogart, C., Burnett, M., Bellamy, R., Rector, K., Fleming, S.D.: How programmers debug, revisited: An information foraging theory perspective. IEEE Transactions on Software Engineering **39**(2), 197–215 (Feb 2013). https://doi.org/10.1109/TSE.2010.111
12. Li, J.J., Horgan, R.: xsuds-sdl: A tool for diagnosis and understanding software specifications (2004)
13. Mock, M., Gergeleit, M., Nett, E.: A lightweight awareness service for industrial environments [distributed software]. In: Proceedings of the Thirtieth Hawaii International Conference on System Sciences. vol. 1, pp. 433–442 vol.1 (Jan 1997). https://doi.org/10.1109/HICSS.1997.667297
14. Myers, G.J., Badgett, T., Sandler, C.: The Art of Software Testing. John Wiley & Sons, Inc., New Jersey, 3a edn. (2012)
15. de Paiva Guimarães, M., Dias, D.C., Gnecco, B.B., Martins, V.F., Contri, L.F.: Development and debugging of distributed virtual reality applications. In: 2013 8th Iberian Conference on Information Systems and Technologies (CISTI). pp. 1–6 (June 2013)
16. Perez, A., Abreu, R.: Cues for scent intensification in debugging. IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) (2013)
17. Reiss, S.P.: The challenge of helping the programmer during debugging. Second IEEE Working Conference on Software Visualization (VISSOFT) (2014)
18. Roehm, T., Gurbanova, N., Bruegge, B., Joubert, C., Maalej, W.: Monitoring user interactions for supporting failure reproduction. In: 2013 21st International Conference on Program Comprehension (ICPC). pp. 73–82 (May 2013). https://doi.org/10.1109/ICPC.2013.6613835
19. Sasso, T.D., Lanza, M.: In*bug: Visual analytics of bug repositories. IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE) (2014)
20. Seghir, M.N., Kroening, D.: A visual studio plug-in for cprover. In: 2013 3rd International Workshop on Developing Tools as Plug-Ins (TOPI). pp. 43–48 (May 2013). https://doi.org/10.1109/TOPI.2013.6597193
21. SOMMERVILLE, L.: Software Engineering. Pearson Education, São Paulo, 8a edn. (2007)
22. Tanimoto, S.L.: A perspective on the evolution of live programming. In: 2013 1st International Workshop on Live Programming (LIVE). pp. 31–34 (May 2013). https://doi.org/10.1109/LIVE.2013.6617346
23. Ueda, Y., Kamiya, T., Kusumoto, S., Inoue, K.: Gemini: maintenance support environment based on code clone analysis. In: Proceedings Eighth IEEE Symposium on Software Metrics. pp. 67–76 (2002). https://doi.org/10.1109/METRIC.2002.1011326
24. Wesonga, S., Mercer, E.G., Rungta, N.: Guided test visualization: Making sense of errors in concurrent programs. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). pp. 624–627 (Nov 2011). https://doi.org/10.1109/ASE.2011.6100141
25. Zhao, C., Zhang, K., Hao, J., Wong, W.E.: Visualizing multiple program executions to assist behaviour verification. Third IEEE International Conference on Secure Software Integration and Reliability Improvement (2009)