A ARTE DA PESQUISA BIBLIOGRÁFICA NA BUSCA DO CONHECIMENTO SOBRE A ADOÇÃO DE BOAS PRÁTICAS NO DESENVOLVIMENTO DE SOFTWARE

THE ART OF BIBLIOGRAPHIC RESEARCH IN THE SEARCH FOR KNOWLEDGE ABOUT ADOPTING GOOD PRACTICES IN SOFTWARE DEVELOPMENT

Luiz Felipe Muniz Regis[1]

Roberto F. de Oliveira[2]

Resumo: O presente trabalho foi desenvolvido durante uma iniciação científica e teve como contexto a melhora da qualidade estrutural do código fonte. Anomalias de código são trechos de código fonte com problemas estruturais, que dificultam as atividades de manutenção e evolução de um *software*. As anomalias devem ser detectadas e removidas o mais cedo possível, pois podem degradar a estrutura de um *software*. A partir disso, o objetivo foi levantar as práticas adotadas para detecção de anomalias de código. Para alcançar esse objetivo foi realizado um mapeamento sistemático, que contou com as seguintes etapas: (1) orientações ao aluno de iniciação científica; (2) planejamento da pesquisa; (3) coleta e análise dos artigos e (4) apresentação dos resultados. De modo geral, nossos resultados demonstram que as práticas mais adotadas para detecção de anomalias de código foram ferramentas, métricas e técnicas. Os dados levantados podem auxiliar os docentes na escolha das práticas a serem estimuladas durante a formação dos futuros desenvolvedores de *software*. Também podem ser utilizados como fonte de consulta aos pesquisadores sobre como se encontra o atual estado da arte em relação às práticas de detecção de anomalias de código.

Palavras-chave: Desenvolvimento de software. Qualidade estrutural. Iniciação científica.

Abstract: *The present work was developed during a scientific initiation and had as context the improvement of the structural quality of the source code. Code smells are stretches of source code with structural problems that make it difficult to maintain and evolve a software. The smells should be detected and removed as soon as possible, because they can degrade the software structure. From this, the objective was to survey the practices adopted for detection of code smells. To achieve this objective, a systematic mapping was carried out, which included the following steps: (1) orientation to the students of scientific initiation; (2) research planning; (3) collection and analysis of articles; and (4) presentation of the results. Overall, our results demonstrate that the most widely adopted code smells detection practices were tools, metrics and techniques. The data collected can support the teachers choosing the practices to be stimulated during the academic training of the future software developers. They can also be used as a source for the researchers to inquire about current state of art regarding code smell detection practices.*

Keywords: *Software development. Structural quality. Scientific research.*

[1]Bacharelando em Sistemas de Informação, UEG-Posse, luipodm@gmail.com.
[2]Professor Orientador, Pós-Doutor em Informática – PUC-RIO, Docente de Ensino Superior – RTIDP-Universidade Estadual de Goiás. E-mail para contato: roberto.oliveira@ueg.br.

1. INTRODUÇÃO

A compreensão do código fonte de um *software* é essencial para sua manutenção e evolução (ABBES *et al*., 2011). Visto que as atividades de manutenção demandam muito esforço e custo, logo se faz necessária uma avaliação da qualidade do código (YAMASHITA e MOONEN, 2013; PIGOSKI, 1996). Beck e Fowler (1999) definiram 22 padrões de estruturas de código com baixa qualidade, denominados anomalias de código. As anomalias de código são indicadores de que o código fonte apresenta problemas que dificultam a manutenção e evolução do *software* (BECK e FOWLER, 1999). Elas podem afetar diferentes elementos de um *software* como métodos, classes e pacotes.

Anomalias de código não indicam erros de compilação ou de execução no código, mas sim baixa capacidade de ser compreendido, reutilizado, mantido e evoluído (GARCIA *et al*., 2009; OLBRICH *et al*., 2009). Uma enorme quantidade de anomalias de código num sistema pode degradar a sua estrutura a ponto de ser totalmente reestruturado (MACIA, *et al.*, 2012; OLIVEIRA, *et al.*, 2016). Logo se faz necessária a identificação e remoção das anomalias de código (BECK e FOWLER, 1999).

Estudos prévios apontam que alunos de graduação em cursos de computação podem introduzir anomalias em seus programas assim que começam a aprender programação (HERMANS e AIVALOGLOU, 2016). É possível que essa observação seja característica inerente ao ensino de linguagens de programação, ligada ao seu conteúdo árido, determinado por uma lógica inflexível. Entretanto, pode ser que os motivos estejam ligados a uma metodologia de ensino inadequada. Nesse sentido, uma catalogação bibliografia, pode ajudar os docentes a localizar e identificar quais as melhores práticas a serem adotadas em disciplinas de programação a fim de formar profissionais com competências para construção de *software* com elevado índice de qualidade estrutural.

2. OBJETIVOS

O presente trabalho é parte do projeto de pesquisa intitulado: **Uma investigação sobre o impacto de competências na qualidade estrutural do *software***. Assim, nós realizamos um mapeamento sistemático sobre a adoção de boas práticas para construção de *software* legível, fácil de manter e evoluir. Mapeamento sistemático é uma forma de identificar, avaliar e interpretar todos os trabalhos disponíveis para uma questão de pesquisa

particular. Umas das razões para a realização de revisões sistemáticas é que esta sumariza as evidências existentes em relação a um tratamento ou tecnologia. (KITCHENHAM, 2004).

Nós descrevemos o objetivo geral desse trabalho segundo o paradigma *Goal-Question-Metric* (BASILI *et al*., 1994), Portanto, nós buscamos: **Analisar** publicações científicas através de um estudo baseado em mapeamento sistemático; **Com o propósito de** identificar e caracterizar habilidade e aptidões necessárias para desenvolvimento de *software* com qualidade estrutural; **Em relação** à definição e uso de boas práticas no desenvolvimento de *software*; **Do ponto de vista** de pesquisadores e organizações de *software*.

Espera-se que esse trabalho, possa ser usado por pesquisadores como uma fonte de consulta sobre como encontra-se o estado da arte em relação ao desenvolvimento de *software* com qualidades estruturais. E por docentes de modo a avançar o conhecimento sobre as principais competências para a formação de alunos de graduação nos cursos de computação no sentido de auxiliá-los no desenvolvimento de *software* legível, fácil de manter e evoluir.

2.1 OBJETIVOS ESPECÍFICOS

Para realizar o objetivo principal desse trabalho, procuramos atingir os seguintes objetivos específicos: (I) identificação das principais práticas adotadas na academia e na indústria para o desenvolvimento de *software* com elevados índices de qualidade estrutural; e (II) uma catalogação bibliográfica dessas práticas.

3. METODOLOGIA

Neste trabalho, nós optamos por realizamos uma pesquisa exploratória. Esse tipo de pesquisa tem como objetivo proporcionar maior familiaridade com o problema, com vistas a torna-lo mais explícito (GIL, 2002). Na **Figura 1** apresentamos as quatro fases da metodologia de pesquisa baseada no nosso estudo exploratório:

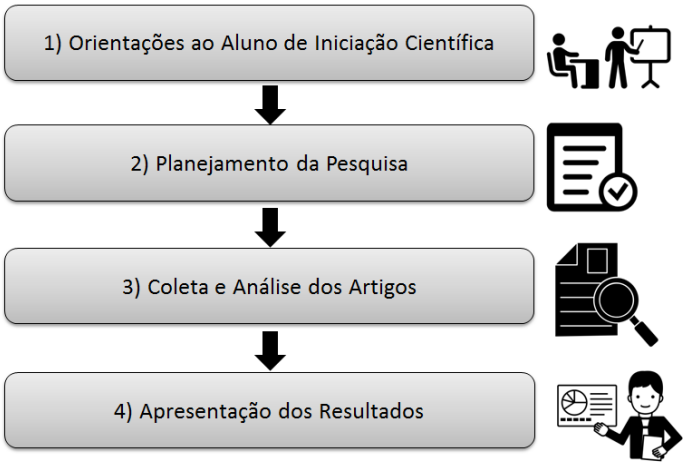

Figura 1: Fases do Mapeamento Sistemático

A **primeira fase** compreendeu a uma série de orientações ao aluno de iniciação científica por meio de exposições teóricas e atividades práticas por parte do orientador. Nessas exposições, o aluno de iniciação científica teve direcionamentos de como realizar pesquisas, como identificar e reportar resultados relevantes entre outros aspectos. A **segunda fase** correspondeu ao planejamento do mapeamento sistemático. Nessa fase, nós determinamos o escopo do mapeamento sistemático, o idioma a ser considerado, os termos utilizados como filtros para a pesquisa, a *string* de busca adotada, os critérios de seleção e exclusão dos artigos retornados pela *string* de busca entre outros aspectos. A **terceira fase** correspondeu à coleta, classificação e análise dos artigos científicos. A **quarta fase** correspondeu à apresentação dos resultados e orientações junto à comunidade acadêmica por meio de pôsteres. Destaca-se que essa fase ainda se encontra em andamento, conforme apresentado nas Seções 4 e 5.

4. RESULTADOS e DISCUSSÃO

Como primeiro resultado, destacamos a capacitação do aluno de iniciação científica. Essa capacitação ocorreu em paralelo com a segunda e terceira fases da metodologia dessa pesquisa. Onde o aluno adquiriu conhecimento prático através do planejamento da pesquisa e da coleta e análise dos artigos. A *string* de busca definida foi baseada nos seguintes termos: "*adoption*", "*detect*" e "*code smells*". Ela foi aplicada nas bases de dados *Google Scholar* e *Scopus*, que foram escolhidas por possuírem um grande acervo de artigos indexados. A *string* retornou 357 artigos, que foram adicionados em uma planilha eletrônica e passaram pelo processo de filtragem. Após o processo de filtragem, restaram 175 trabalhos, que foram

analisados detalhadamente. A partir dessa análise identificamos diversos trabalhos que abordam uma ou mais práticas para a detecção de anomalias de código.

Os trabalhos levantados e as práticas identificadas pelo mapeamento sistemático foram apresentados para a comunidade acadêmica de duas formas: (1) através da publicação de um resumo expandido no V Congresso de Ensino, Pesquisa e Extensão (CEPE), nele conteve os resultados preliminares obtidos a partir da primeira filtragem; (2) através da confecção de um artigo completo para a Revista Mundi Engenharia, Tecnologia e Gestão, o qual será submetido no segundo semestre de 2019; (3) através da confecção do Trabalho de Curso a ser apresentado no segundo semestre de 2019 no curso de Sistemas de Informação da UEG Campus Posse A **Figura 2** demonstra a participação do aluno de iniciação científica no V CEPE, apresentando os resultados parciais do mapeamento sistemático.

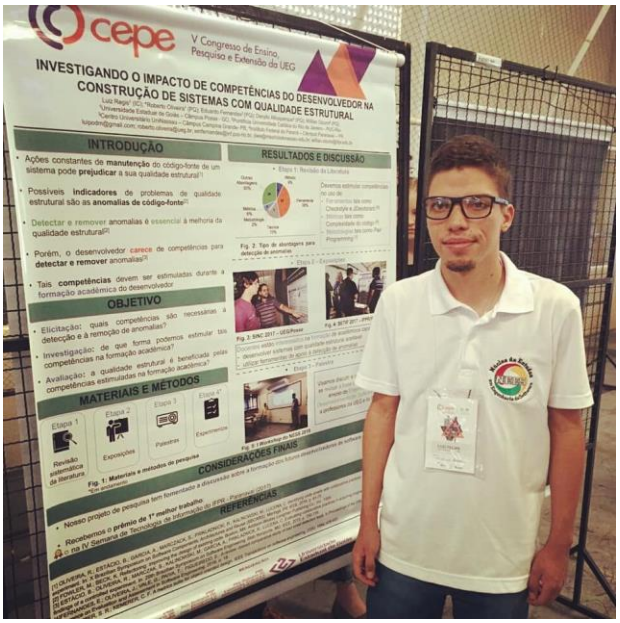

Figura 2: Aluno de Iniciação Científica no CEPE

O mapeamento sistemático coletou trabalhos publicados no período de 2008 e 2018 para não haver problemas com desatualização. Sem contar que muito da literatura relevante anterior ao período escolhido foi mencionado nos artigos do mapeamento. A **Figura 3** apresenta esses trabalhos, organizados por ano de publicação. Como pode ser observado, muitos pesquisadores demonstraram interesse em propor e aplicar ao menos uma prática para apoiar a identificação das anomalias de código. O ano de 2017 contou com o maior número de publicações (36), enquanto o ano de 2008 foi o que menos teve trabalhos (5).

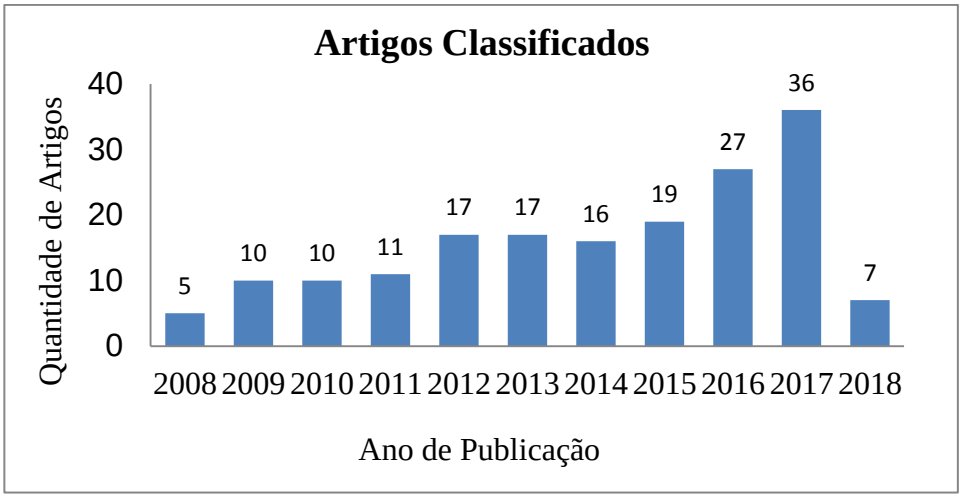

Figura 3: Artigos que abordaram práticas para detecção de anomalias de código

É possível perceber que o número de trabalhos só aumentou a cada ano, tendo um declínio entre os anos de 2013 e 2014, mas em 2015 em diante continuou a crescer. A queda alta entre os anos de 2017 e 2018 no número de publicações foi devido a extração dos artigos ter sido realizada no primeiro semestre de 2018. Portanto, nossa primeira descoberta é: **A tendência de pesquisas envolvendo práticas que auxiliem na detecção de anomalias de código é crescer**, pois com esses trabalhos pode ser inferido que há uma enorme preocupação da comunidade acadêmica em criar e manter código com boa qualidade.

A próxima descoberta é relacionada com as práticas para detecção de anomalias de códigos adotadas nesses trabalhos. A **Figura 4** apresenta as 6 práticas propostas para detecção de anomalias de código: ferramenta, métrica, técnica, método, metodologia e outras práticas.

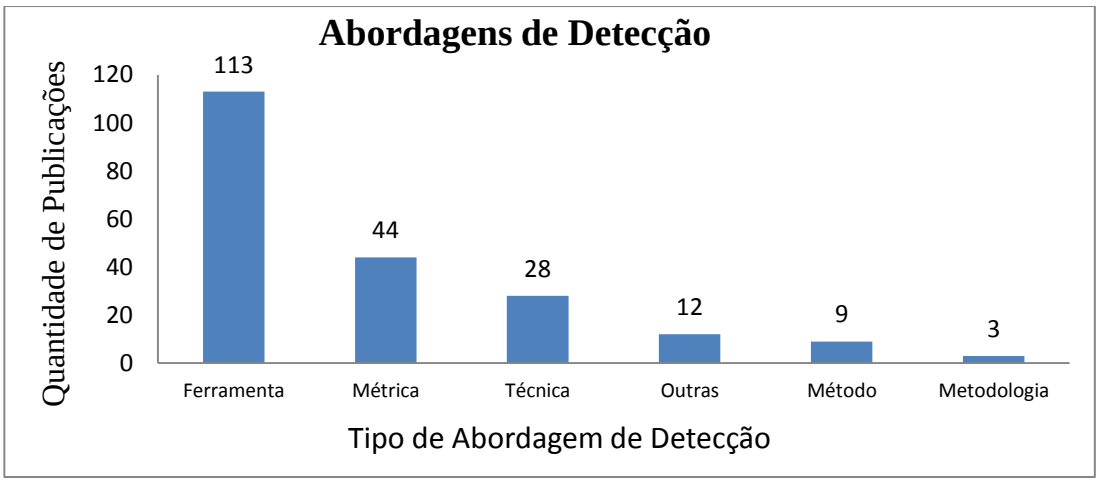

Figura 4: Tipos de práticas adotadas para detecção de anomalias de código

Dos estudos analisados, observamos que a prática mais adotada foi a detecção de anomalias com auxílio de ferramentas, um total de 113 trabalhos, dentre esses trabalhos destacamos os de Palomba *et al*. (2017), Liu *et al*. (2012) e Soh *et al*. (2016). O trabalho de Palomba *et al.* (2017) utilizou as ferramentas Decor e JDeodorant para identificar as anomalias Classe de Deus, Inveja de Recurso, Método Longo e entre outras. No trabalho de Liu *et al*. (2012) foram adotadas as ferramentas Metrics, PMD e JDeodorant para detectar as anomalias Método Longo, Código Duplicado e Inveja de Recurso e entre outras.

O segundo tipo de prática mais adotada foi métrica, utilizado em 44 estudos. Alguns trabalhos propunham uma nova métrica como o de Zaparanuks e Hauswirth (2011), mas a maioria usou métricas existentes como o trabalho de Olbrich *et al.* (2009). O trabalho de Olbrich *et al*.(2009) buscou detectar os odores Classe de Deus e Cirurgia de Espingarda utilizando métricas propostas por Marinescu (2004). O terceiro tipo de prática mais adotada foi técnica com 28 estudos adotando o seu uso, dentre esses estudos estão os de Moha *et al*. (2008) e Bakota (2011). O trabalho de Moha *et al*.(2008) apresenta uma técnica de geração de algoritmos para detectar anomalias de código. As práticas menos frequentes foram do tipo: Outras (12), Método (9) e Metodologia (3). Baseados nesses resultados chegamos ao nosso segundo achado: **é fundamental inserir o uso de ferramentas de detecção de anomalias de código nas bases curriculares dos cursos de computação**. Essa inclusão é importante pois despertará o interesse do aluno na identificação de anomalias de código desde cedo.

A **Figura 5** apresenta as 7 ferramentas mais aplicadas na detecção de anomalias de código: *JDeodorant, PMD, DECOR, InFusion, InCode, iPlasma* e *Borland Together.*

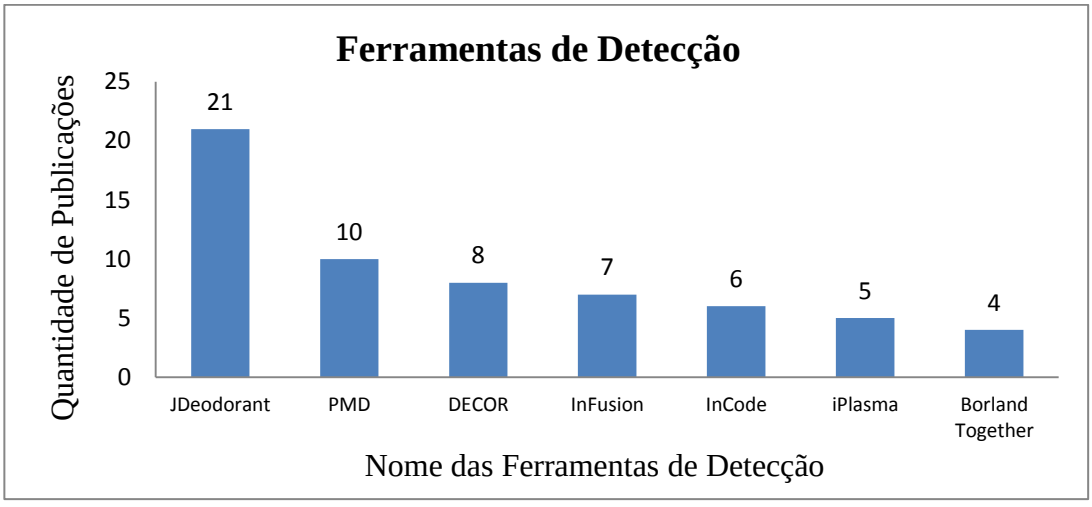

Figura 5: As ferramentas mais adotadas para detecção de anomalias de código

A ferramenta mais utilizada foi a *JDeodorant* (21), alguns dos trabalhos que a utilizaram foram os de Chen, Liu e Li (2018) e Lozano, Mens e Portugal (2015). Essa ferramenta é capaz de realizar a detecção de anomalias como Inveja de Recurso e Classe de Deus. Ela também auxilia na remoção da anomalia de código. Dez estudos reportaram o uso da ferramenta *PMD,* dentre eles o de Palma *et al.* (2014) e Liu *et al.* (2012). Ela realiza a detecção de anomalias como Código Duplicado, Classe de Deus, Método Longo e Lista de Parâmetros Longa. A ferramenta *InFusion* (7), aplicada nos trabalhos de Yoshida et al. (2016) e Shae-Lim, Hayashi e Saeki (2017), pode detectar as anomalias Classe de Deus, Inveja de Recurso e Código Duplicado. As ferramentas *Borland Together* e *iPlasma* foram adotadas em cinco estudos. As ferramentas são interessantes para ser aplicada no ensino da detecção de anomalias de código devido a praticidade no seu uso. Porém é importante fazer uma análise dessa abordagem para identificar as barreirais, benefícios e outros aspectos de sua adoção.

5. CONSIDERAÇÕES FINAIS

Este trabalho apresentou um mapeamento sistemático das práticas usadas na literatura para a detecção de anomalias. Esse mapeamento poderá ser utilizado como fonte de consulta para pesquisadores, entender as práticas adotadas para construir sistemas com qualidade estrutural. Também é para os professores saberem quais são as competências necessárias ser ensinadas aos alunos de modo que possam construir sistemas fáceis de manter e evoluir. Através desse estudo vimos que as ferramentas são interessantes para apoiar o ensino da detecção de anomalias de código, devido a praticidade. No relatório parcial de Iniciação Científica utilizamos dados somente da base de dados *Google Scholar*, não tendo sido feita a coleta dos artigos na base de dados *Scopus*. O resultado final do mapeamento sistemático será submetido como artigo completo na **Revista Mundi Engenharia, Tecnologia e Gestão** e também será base do Trabalho de curso do referido aluno de científica.

6. AGRADECIMENTOS

Agradeço aos meus familiares, ao Prof. Dr. Roberto Oliveira (Orientador) e aos demais membros do Núcleo de Estudos de Engenharia de *Software* (NEES), pelo apoio prestado durante a execução do projeto. Agradeço também ao programa de iniciação científica da Universidade Estadual de Goiás (UEG).

7. REFERÊNCIAS

ABBES, Marwen et al. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension. In: **2011 15th European Conference on Software Maintenance and Reengineering**. IEEE, 2011. p. 181-190.

BAKOTA, Tibor. Tracking the evolution of code clones. In: **International Conference on Current Trends in Theory and Practice of Computer Science**. Springer, Berlin, Heidelberg, 2011. p. 86-98.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. **Goal Question Metric Paradigm**. Encyclopedia of Software Engineering. [S.l.]: Marciniak, J.J. 1994. p. 528-532.

BECK, Kent; FOWLER, Martin; BECK, Grandma. Bad smells in code. **Refactoring: Improving the design of existing code**, p. 75-88, 1999.

BROWN, William H. et al. **AntiPatterns: refactoring software, architectures, and projects in crisis**. John Wiley & Sons, Inc., 1998.

CHEN, Woei-Kae; LIU, Chien-Hung; LI, Bo-Hong. A Feature Envy Detection Method Based on Dataflow Analysis. In: **2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)**. IEEE, 2018. p. 14-19.

GARCIA, Joshua et al. Identifying architectural bad smells. In: **2009 13th European Conference on Software Maintenance and Reengineering**. IEEE, 2009. p. 255-258.

GIL, Antonio Carlos. Como elaborar projetos de pesquisa. **São Paulo**, v. 5, n. 61, p. 16-17, 2002.

HERMANS, Felienne; AIVALOGLOU, Efthimia. Do code smells hamper novice programming? A controlled experiment on Scratch programs. In: **2016 IEEE 24th International Conference on Program Comprehension (ICPC)**. IEEE, 2016. p. 1-10.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. Joint Technical Report Software Engineering Group. Australia: [s.n.]. 2004. p. 1-33.

LIU, Hui et al. Schedule of bad smell detection and resolution: A new way to save effort. **IEEE transactions on Software Engineering**, v. 38, n. 1, p. 220-235, 2011.

LOZANO, Angela; MENS, Kim; PORTUGAL, Jawira. Analyzing code evolution to uncover relations. In: **2015 IEEE 2nd International Workshop on Patterns Promotion and Anti-patterns Prevention (PPAP)**. IEEE, 2015. p. 1-4.

MACIA, Isela et al. On the relevance of code anomalies for identifying architecture degradation symptoms. In: **2012 16th European Conference on Software Maintenance and Reengineering**. IEEE, 2012. p. 277-286.

MARINESCU, Radu. Detection strategies: Metrics-based rules for detecting design flaws. In: **20th IEEE International Conference on Software Maintenance, 2004. Proceedings.** IEEE, 2004. p. 350-359.

MOHA, Naouel et al. A domain analysis to specify design defects and generate detection algorithms. In: **international conference on Fundamental approaches to software engineering**. Springer, Berlin, Heidelberg, 2008. p. 276-291.

OLBRICH, Steffen et al. The evolution and impact of code smells: A case study of two open source systems. In: **2009 3rd international symposium on empirical software engineering and measurement.** IEEE, 2009. p. 390-400.

OLIVEIRA, Roberto et al. Identifying code smells with collaborative practices: A controlled experiment. In: **2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)**. IEEE, 2016. p. 61-70.

PALOMBA, Fabio et al. The scent of a smell: An extensive comparison between textual and structural smells. **IEEE Transactions on Software Engineering**, v. 44, n. 10, p. 977-1000, 2017.

SAE-LIM, Natthawute; HAYASHI, Shinpei; SAEKI, Motoshi. How do developers select and prioritize code smells? a preliminary study. In: **2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)**. IEEE, 2017. p. 484-488.

SOH, Zéphyrin et al. Do code smells impact the effort of different maintenance programming activities?. In: **2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)**. IEEE, 2016. p. 393-402.

PALMA, Francis et al. Investigating the change-proneness of service patterns and antipatterns. In: **2014 IEEE 7th International Conference on Service-Oriented Computing and Applications**. IEEE, 2014. p. 1-8.

PIGOSKI, Thomas M. **Practical software maintenance: best practices for managing your software investment**. Wiley Publishing, 1996.

YAMASHITA, Aiko; MOONEN, Leon. To what extent can maintenance problems be predicted by code smell detection?–An empirical study. **Information and Software Technology**, v. 55, n. 12, p. 2223-2242, 2013.

YOSHIDA, Norihiro et al. Revisiting the relationship between code smells and refactoring. In: **2016 IEEE 24th International Conference on Program Comprehension (ICPC)**. IEEE, 2016. p. 1-4.

ZAPARANUKS, Dmitrijs; HAUSWIRTH, Matthias. The beauty and the beast: separating design from algorithm. In: **European Conference on Object-Oriented Programming**. Springer, Berlin, Heidelberg, 2011. p. 27-51.