Entre Ciencia e Ingeniería, vol. 17, no. 33, julio-diciembre de 2023, páginas 9-14. DOI: https://doi.org10.31908/19098367.2846
ISSN 1909-8367 (Impreso), ISSN 2539-4169 (En línea)

Static Code Analysis: A Tree of Science Review

Análisis de código estático: una revisión del árbol de la ciencia

G. A. Ruiz, S. Robledo y H. H. Morales

Recibido: octubre 8 de 2022 – Aceptado: julio 5 de 2023

***Resumen*—El análisis de código estático (SA) es el proceso de encontrar vulnerabilidades en el software. Este proceso se ha vuelto popular y una de las fases más evaluadas en el proceso de integración continua de software. Sin embargo, la literatura se encuentra dispersa en diferentes propuestas y faltan investigaciones que muestran las principales contribuciones y aplicaciones a este tema. El propósito de este artículo es identificar las principales contribuciones conceptuales de SA utilizando el algoritmo Tree of Science. Los resultados muestran tres ramas principales de esta área: aprendizaje automático para la detección de olores, técnicas de clasificación accionables y herramientas de alerta técnica. La Inteligencia Artificial ha estado transformando SA y los programadores tendrán acceso a herramientas más sofisticadas.**

***Palabras clave*— análisis estático, alertas, errores, defectos, advertencias, código fallido.**

***Abstract*— Static Code Analysis (SA) is the process of finding vulnerabilities in software. This process has become popular and one of the most evaluated phases in the process of continuous integration of software. However, the literature is spread over different proposals and there is a lack of research that shows the main contributions and applications to this topic. The purpose of this paper is to identify the main conceptual contributions of SA using the Tree of Science algorithm. The results show three main branches of this area: machine learning for smell detection, actionable ranking techniques, and Technical alert tools. Artificial Intelligence has been transforming SA and programmers will have access to more sophisticated tools.**

***Keywords*—static analysis, alert, bug, defect, warning, fault code.**

I. Introduction

FINDING the best practices in software coding is an important and demanding task, only improving the quality of the code is a tedious process. For example, developers can expend between 35-50% of their time just debugging their code *(https://thenewstack.io/the-time-travel-method-of-debugging-software/)* which can cost billions of dollars *(https://metabob.com/blog-articles/shifting-left.html),* many companies prioritize speed or quantity over quality *(https://www.knowledgetrain.co.uk/agile/agile-project-management/agile-project-management-course/agile-principles).*

The software development process is an intellectual activity, and time is the most important assessment to calculate the cost of a software product. Companies take seriously the time variable in software production; therefore, they are continuously looking for ways to reduce this variable to improve the profits of the organization. Understanding how the debugging process could be improved by reducing the time spent from a scientific perspective is important to help companies to be more productive.

There are a few papers that explain the different proposals in SA. For example, Heckman and Williams [1] present a well-detailed analysis of the identification techniques in SA. Also, Kaur et al. [2] identified different areas of code smell detection using several machine learning and hybrid techniques. The authors show that hybrid approaches present better results in comparison to machine learning. Also, Al-Shaaby et al. [3] wrote a systematic literature review about smell code detection with machine learning techniques. Azeem et al. [4] present a literature review of code smell with machine learning. Akremi [5] provided recent efforts in static analysis tools to classify different false positive outcomes. However, there is a missing paper that shows the evolution of this important topic with its subfields.

The purpose of this paper is to identify the main contributions of SA in academics. We applied the Tree of Science (ToS) algorithm from a Scopus search to show the evolution of SA from a tree perspective [6]. This paper presents the evolution of SA with a tree metaphor; papers in roots represent the classics, the trunk the structurals, and the

[1] Producto derivado del proyecto de investigación “Análisis cienciométrico de la producción científica de la universidad católica luis amigó”, apoyado por la Universidad Catolica Luis Amigó, a través de la Especialización en Big Data e Inteligencia de Negocios

G. A. Ruiz, Universidad Catolica Luis Amigó, Medellín, Colombia, email: gustavo.ruizmo@amigo.edu.co.

S. Robledo, Universidad Catolica Luis Amigó, Manizales, Colombia, email: gustavo.ruizmo@amigo.edu.co.

H. H. Morales, Universidad Catolica Luis Amigó, Medellín, Colombia, email: huber.moralesmo@amigo.edu.co.

Cómo citar este artículo: G. A. Ruiz, S. Robledo y H. Morales, Static Code Analysis: A Tree of Science Review, Entre Ciencia e Ingeniería, vol. 17, no. 34, pp. 9-14, julio - diciembre 2023.
DOI:https://doi.org/10.31908/19098367.2846.

branches present the current subfields [7]. The platform to create the ToS of SA was built using R and Shiny apps and Works only with Scopus files *(https://coreofscience.shinyapps.io/scientometrics/)*.

The results show a growing development of SA and show the importance of the subject in academic literature. SA has been transformed since the first proposal and, today, Artificial Intelligence (AI) has become a relevant issue in this field because this technology continues to learn from possible errors and bad practices. This is a cyclical exercise that allows generating a specialized tool to alert when it is incurring a failure, in the future can be the entrance for errors that can be costly to solve after a production start-up or the implementation of processes that depend on a code fragment with sensitive vulnerabilities.

The rest of the paper is split into three parts. The first part presents the methodology of ToS, and the second part shows the main results with a tree metaphor. Finally, we explain the main contributions of SA to academic literature.

II. Methodology

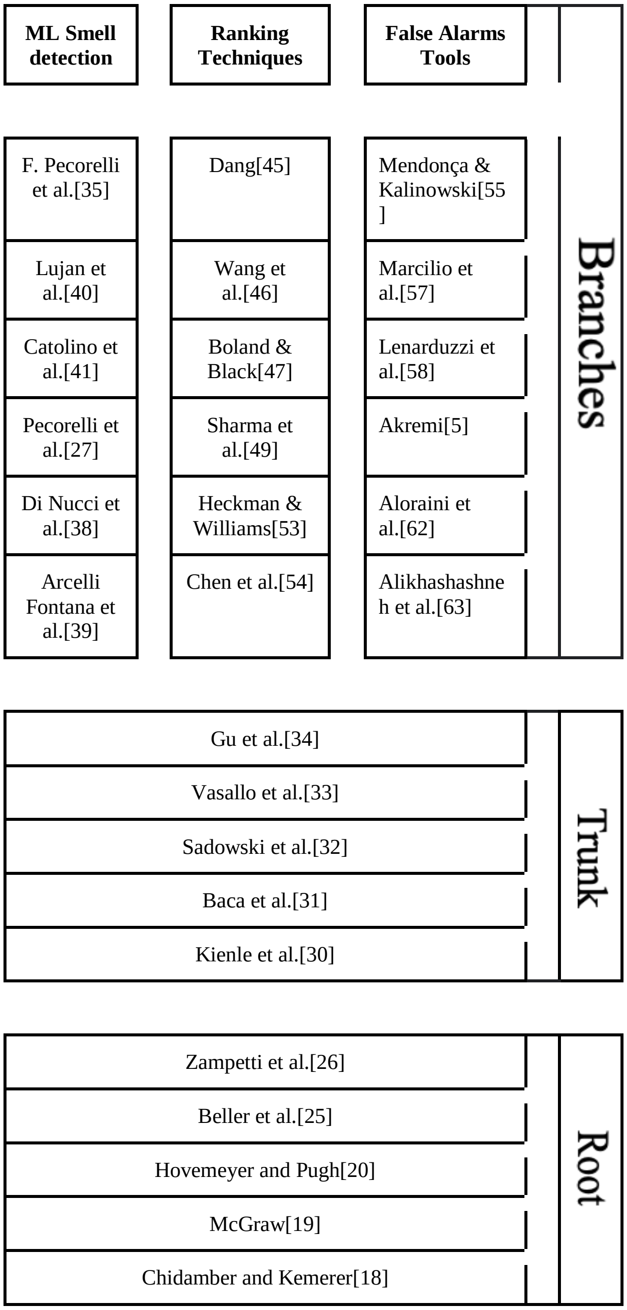
We used the Scopus database to identify the main literature on SA because it is recognized as one of the most important and one of the biggest databases [8]. The words searched were: “static analysis” in the title and “alert or bug or defect or fault or warning” in the title, abstract, and keywords. We had 199 results from the search without filtering by year or document type. Next, the seed (the 199 registers) was uploaded to the shiny application to identify the ToS of our topic. ToS creates a citation network and then applies the SAP algorithm to identify the relevant papers [6], [9]. ToS has been applied to identify relevant literature in management [10]–[12], psychology [13], education [14], health [15], and engineering [16]. The explanation of the ToS diffusion is found in Eggers et al. [17].

III. Results

A. Root

ToS algorithm presents the papers in a tree shape that lets us discover the SA topic's evolution (see Figure 1). The first paper in the roots is the study of Chidamber and Kemerer [18], who implemented six metrics to evaluate software development. Chess and McGraw[19] were more specific in SA, they show how security analysis could be applied in SA. Hovemeyer and Pugh [20] proposed a system to automatically identify bugs that makes this task easier. Heckman [21] identified the bottleneck of the SA programs, they identified many false positives, therefore, she proposed a ranking algorithm to help developers select the most important bugs in their code. Another solution to this problem is proposed by Ruthruff et al. [22], who suggested a logistic regression to identify the most relevant bugs. However, Bessey et al. [23] show the pros and cons of SA software, they have been in the industry for many years and shared their positive and negative experiences. Johnson et al. [24] conducted several interviews with developers and identified the same problems with false positives. Beller et al. [25] presented an analysis of different tools for SA and show that they are widely used and these tools have minor modifications through time. Finally, Zampetti et al. [26] present the benefits of SA tools.

TABLE I.
TREE OF SCIENCE OF STATIC CODE ANALYSIS.



ML Smell detection	**Ranking Techniques**	**False Alarms Tools**	
F. Pecorelli et al.[35]	Dang[45]	Mendonça & Kalinowski[55]	Branches
Lujan et al.[40]	Wang et al.[46]	Marcilio et al.[57]	
Catolino et al.[41]	Boland & Black[47]	Lenarduzzi et al.[58]	
Pecorelli et al.[27]	Sharma et al.[49]	Akremi[5]	
Di Nucci et al.[38]	Heckman & Williams[53]	Aloraini et al.[62]	
Arcelli Fontana et al.[39]	Chen et al.[54]	Alikhashashneh et al.[63]	

Gu et al.[34]	Trunk
Vasallo et al.[33]	
Sadowski et al.[32]	
Baca et al.[31]	
Kienle et al.[30]	

Zampetti et al.[26]	Root
Beller et al.[25]	
Hovemeyer and Pugh[20]	
McGraw[19]	
Chidamber and Kemerer[18]	

B. Trunk

Documents in the trunk show the evolution of different features of static code analysis. For example, Hallem et al. [27] proposed the metal system to identify bugs in the code.

Zheng et al. [28] show that automated static analysis is beneficial for companies. Ayewah et al. [29] show the perception of the effectiveness of FingBugs. Also, Kienle et al. [30] explored how the industry could implement software to automate static analysis. Baca et al. [31] focus their attention on security analysis in a case study. Sadowski et al. [32] highlight the importance of developer workflow integration in static analysis tool adoption. Vasallo et al. [33] proposed that Automatic Static Analysis Tools (ASA) are applied differently depending on the context. Finally, Gu et al. [34] expose the challenge of SA in large projects and proposed BigSpa for this.

C. *Branch 1: Machine Learning for Smell Detection*

The first branch is code smell detection. Code smells are "symptoms that something may be wrong in software systems that can cause complications in maintaining software quality" [2]. Machine Learning (ML) is the most popular strategy to identify these types of issues. For example, Pecorelli et al. [35] investigate the impact of using ML approaches in code smell detection. They found that balancing techniques are still in their infancy leading to low accuracy outcomes. However, the ML proposals suffer from low accuracy, and comparisons from random models present similar performances [36]. Also, Pecorelli et al. [37] compare the performance of ML models resulting in low values of accuracy. Di Nucci et al. [38] replicated a study with different datasets with more types of smells. Arcelli Fontana et al. [39] tested 16 different ML algorithms and showed that high performances are archived by random forest; also, Lujan et al. [40] found an important increase when they compared several ML tools for code smell. Catolino et al. [41] proposed an intensity index metric based on ML models to identify the severity of a smell code. Also, Pecorelli et al. [42] proposed an ML approach to rank code smells. ML can detect code smells [43]. Das et al. [44] proposed a model to detect two types of smells using a deep learning approach.

D. Branch 2: Actionable Ranking Techniques

The ranking techniques branch is the oldest one in the tree, the newest paper is from 2020 and the next one is from 2013. Even though this branch has 87 papers, it only has 10 papers from the last 10 years.

Static analysis tools show software warnings; however, the results show too many alerts overwhelming the developers. Dang [45] proposed identifying the most relevant warnings in Java and C++. Wang et al. [46] used support vector machines and Naïve Bayes to identify and classify new bugs. Boland & Black [47] proposed Juliet for testing compromising important warnings in C/C++ and Java. Allier et al. [48] proposed a framework of six algorithms for ranking techniques. They show that algorithms based on past alerts perform better than linear regression or Bayesian network techniques. Sharma et al. [49] assessed ML techniques to prioritize bugs in SA and the ML performance was above 70%. Shen et al. [50] presented EFindBugs tool to remove false positives and rank real bugs. EFindBugs is a self-adaptive program that uses the user's feedback to optimize prioritization in Java projects. Liang et al. [51] proposed a training set to identify warnings and automatically label them with 100% of accuracy. Nanda et al. [52] developed Khasiana, an online portal for SA that provides multiple analysis tools for depth analysis, different important warnings, and customized defect reports. Heckman & Williams [53] used ML techniques to classify actionable warnings; they obtain 88-97% accuracy using three to 14 alert features. Chen et al. [54] implemented IntFinder combining static and dynamic analysis providing more accurate bug predictions.

E. Branch 3: Technical alert tools

Mendonça and Mendonça & Kalinowski [55] proposed Pattern-driven maintenance (PDM), a method to create new rules in SA. Serban et al. [56] created SAW-BOT, a bot that proposes fixes for static analysis alerts. Marcilio et al. [57] present SpongeBugs for recommending fixes to developers. Lenarduzzi et al. [58] show the importance of identifying the rules applied in SonarQube because it could reduce fault-proneness. Akremi [5] proposed a categorization of SA techniques to compare different tools. Liu et al. [59] created AVATAR to automate the correctness of bugs, in other words, a patch generator. They showed that AVATAR can fix more than 90% of the bugs. Also, Bavishi et al. [60] created PHOENIX to automatically program repair with high performance. Wyrich & Bogner [61] presented the Refactoring-Bot to remove Java bugs automatically. Aloraini et al. [62] proposed OWASP to automate the detection of vulnerabilities and Alikhashashneh et al. [63] created a new metric to identify errors called F-measure.

IV. DISCUSSIONS

What are the main reasons to find smelly software?

The maintainability of source code is an ongoing task in the field of software development. As long as there are software solutions that fulfill a task through an algorithm, there will come a time when they become obsolete or prone to bugs. This is where continuous maintenance of the source code comes into play. It involves updating and correcting the code to adhere to best practices and prevent system failures. Developers must stay updated with technological trends to enhance system resilience and minimize future errors.

What contributions can be made from Big Data and ML for error detection?

One significant approach is the application of Machine Learning models for various purposes such as identifying common errors, assessing the severity of poorly written code, classifying code smells, automatically identifying vulnerable code, detecting false positives, and improving the accuracy of error prediction. Moreover, ML-based models can propose corrections to code smells and even automatically correct coding errors. These advancements in the field of statistical analysis of software have led to substantial progress, resulting in significant savings in terms of effort and cost for software companies.

What sophisticated tools are there today that can be kept in mind for the SA?

Several tools and projects are available today to support static analysis of code and offer ML-based analysis models. Examples of such tools and projects include SAW-BOT, SpongeBugs, SonarQube, PHOENIX, EFindBugs, Metabob, GitHub Copilot, AVATAR, and OWASP.

V. Conclusions

This study presents a literature review of SA using the ToS metaphor and presents the results in three sections: roots, trunk, and branches. We identified three main branches of SA: ML for smell detection, ranking techniques, and Technical alert tools. ML is becoming increasingly crucial in smell detection but there are still challenges to increasing the accuracy of the results. The ranking techniques branch was a dynamic subfield but due to the difficulty of identifying important actionable bugs and the emerging new techniques to automate the review process, this branch lost activity in recent years. The third branch presents some tools for detecting bugs, some examples are SAW-BOT, SpongeBugs, and SonarQube. One of the cutting-edge techniques identified is the automatization of correctness bugs; for example, PHOENIX is a program that can fix automatically the bugs in a code, in this vein, the programmer will only need to test the software after the automatic detection.

SA is a dynamic research topic and it has evolved fast during the last years; however, emerging technologies such as Metabob *(https://metabob.com/)* and GitHub copilot have proposed big changes in the industry. Therefore, AI is positioning the next generation of SA tools and solutions to bugs and future technical debt there will always be technical debt in a software product because the improvements of implementations or updates are a support stage that will always be necessary, from refactoring by new technologies to the implementation of more productive and simple development patterns, for such reasons it reinforces the theory that measuring the passage of time will always require increasingly sophisticated SA tools that allow for a shorter time to consider possible breaches and recommendations to meet new trends and technological needs that require the maintainability and continuous improvement of the software.

Acknowledgment

This research was partially support by Universidad Católica Luis Amigó under project entitled “Análisis cienciométrico de la producción científica de la Universidad Católica Luis Amigó.” We thank the editor and the two anonymous referees for their help in preparing this paper.

References

[1] S. Heckman and L. Williams, “A systematic literature review of actionable alert identification techniques for automated static code analysis,” *Information and Software Technology*, vol. 53, no. 4, pp. 363–387, Apr. 2011, doi: 10.1016/j.infsof.2010.12.007.

[2] A. Kaur, S. Jain, S. Goel, and G. Dhiman, “A review on machine-learning based code smell detection techniques in object-oriented software system(s),” *Recent Adv. Electr. Electron. Eng. (Former. Recent Pat. Electr. Electron. Eng.)*, vol. 14, no. 3, pp. 290–303, Apr. 2021, doi: 10.2174/2352096513999200922125839.

[3] A. Al-Shaaby, H. Aljamaan, and M. Alshayeb, “Bad Smell Detection Using Machine Learning Techniques: A Systematic Literature Review,” *Arab. J. Sci. Eng.*, vol. 45, no. 4, pp. 2341–2369, Apr. 2020, doi: 10.1007/s13369-019-04311-w.

[4] M. I. Azeem, F. Palomba, L. Shi, and Q. Wang, “Machine learning techniques for code smell detection: A systematic literature review and meta-analysis,” *Information and Software Technology*, vol. 108, pp. 115–138, Apr. 2019, doi: 10.1016/j.infsof.2018.12.009.

[5] A. Akremi, “Software security static analysis false alerts handling approaches,” *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 11, 2021, doi: 10.14569/ijacsa.2021.0121180.

[6] D. S. Valencia-Hernandez, S. Robledo, R. Pinilla, N. D. Duque-Méndez, and G. Olivar-Tost, “SAP Algorithm for Citation Analysis: An improvement to Tree of Science,” *Ing. Inv.*, vol. 40, no. 1, pp. 45–49, Jan. 2020, doi: 10.15446/ing.investig.v40n1.77718.

[7] M. Zuluaga, S. Robledo, G. Osorio-Zuluaga, L. Yathe, Gonzalez, and Taborda, “Metabolomics and pesticides: systematic literature review using graph theory for analysis of references,” *Nova*, vol. 14, no. 25, pp. 121–138, 2016, [Online]. Available: http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1794-24702016000100010

[8] J. A. Moral-Muñoz, E. Herrera-Viedma, A. Santisteban-Espejo, and M. J. Cobo, “Software tools for conducting bibliometric analysis in science: An up-to-date review,” *EPI*, vol. 29, no. 1, Jan. 2020, doi: 10.3145/epi.2020.ene.03.

[9] S. Robledo, A. M. Grisales Aguirre, M. Hughes, and F. Eggers, “‘Hasta la vista, baby’ – will machine learning terminate human literature reviews in entrepreneurship?,” *J. Small Bus. Manage.*, pp. 1–30, Aug. 2021, doi: 10.1080/00472778.2021.1955125.

[10] P. Duque and E. J. D. Oliva, “Tendencias emergentes en la literatura sobre el compromiso del cliente: un análisis bibliométrico,” *Estudios Gerenciales*, pp. 120–132, Mar. 2022, doi: 10.18046/j.estger.2022.162.4528.

[11] Z. B. Torres and C. O. P. Penagos, “Desarrollo tecnológico y de innovación en talleres de confección. Revisión de literatura,” *bol.redipe*, vol. 11, no. 6, pp. 211–224, Jun. 2022, doi: 10.36260/rbr.v11i6.1848.

[12] J. D. G. Castellanos, P. L. D. Hurtado, L. Barahona, and E. Peña, “Marco de referencia y tendencias de investigación de economía colaborativa,” *REC*, vol. 10, no. 16, pp. 267–292, Jan. 2022, doi: 10.53995/23463279.1159.

[13] D. A. Landínez-Martínez, J. F. Arias-Valencia, and A. S. Gómez-Tabares, “Executive Dysfunction in Adolescent with Obesity: A Systematic Review,” *psykhe*, May 2022, doi: 10.7764/psykhe.2020.21727.

[14] J. A. González-Mendoza and M. del M. Calderon-Contreras, “Teletrabajo y sus impactos: una revisión y análisis bibliométrico,” *Aibi revista investig. adm. ing.*, vol. 10, no. 2, Jul. 2022, doi: 10.15649/2346030x.2437.

[15] E. G. Muñoz, R. Fabregat, J. Bacca-Acosta, N. Duque-Méndez, and C. Avila-Garzon, “Augmented Reality, Virtual Reality, and Game Technologies in Ophthalmology Training,” *Information*, vol. 13, no. 5, p. 222, Apr. 2022, doi: 10.3390/info13050222.

[16] J. E. Gutiérrez-Lopera, J. A. Toloza-Rangel, Á. J. Soto-Vergel, O. A. López-Bustamante, and D. Guevara-Ibarra, “VEHÍCULOS TERRESTRES NO TRIPULADOS, SUS APLICACIONES Y TECNOLOGÍAS DE IMPLEMENTACIÓN,” *ingeniare*, no. 30, pp. 47–71, May 2021, doi: 10.18041/1909-2458/ingeniare.30.7925.

[17] F. Eggers, H. Risselada, T. Niemand, and S. Robledo, “Referral campaigns for software startups: The impact of network characteristics on product adoption,” *J. Bus. Res.*, vol. 145, pp. 309–324, Jun. 2022, doi: 10.1016/j.jbusres.2022.03.007.

[18] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” *IEEE Trans. Software Eng.*, vol. 20, no. 6, pp. 476–493, Jun. 1994, doi: 10.1109/32.295895.

[19] B. Chess and G. McGraw, “Static analysis for security,” *IEEE Security Privacy*, vol. 2, no. 6, pp. 76–79, Nov. 2004, doi: 10.1109/MSP.2004.111.

[20] D. Hovemeyer and W. Pugh, “Finding bugs is easy,” *SIGPLAN Not.*, vol. 39, no. 12, pp. 92–106, Dec. 2004, doi: 10.1145/1052883.1052895.

[21] S. S. Heckman, “Adaptively ranking alerts generated from automated static analysis,” *XRDS*, vol. 14, no. 1, pp. 1–11, Dec. 2007, doi: 10.1145/1349332.1349339.

[22] J. R. Ruthruff, J. Penix, J. D. Morgenthaler, S. Elbaum, and G. Rothermel, “Predicting accurate and actionable static analysis warnings: an experimental approach,” in *Proceedings of the 30th international conference on Software engineering*, Leipzig, Germany, May 2008, pp. 341–350. doi: 10.1145/1368088.1368135.

[23] A. Bessey *et al.*, “A few billion lines of code later: using static analysis to find bugs in the real world,” *Commun. ACM*, vol. 53, no. 2, pp. 66–

75, Feb. 2010, doi: 10.1145/1646353.1646374.

[24] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, “Why don’t software developers use static analysis tools to find bugs?,” in *2013 35th International Conference on Software Engineering (ICSE)*, May 2013, pp. 672–681. doi: 10.1109/ICSE.2013.6606613.

[25] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, “Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software,” in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Mar. 2016, vol. 1, pp. 470–481. doi: 10.1109/SANER.2016.105.

[26] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, “How Open Source Projects Use Static Code Analysis Tools in Continuous Integration Pipelines,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 334–344. doi: 10.1109/MSR.2017.2.

[27] S. Hallem, B. Chelf, Y. Xie, and D. Engler, “A system and language for building system-specific, static analyses,” *SIGPLAN Not.*, vol. 37, no. 5, pp. 69–82, May 2002, doi: 10.1145/543552.512539.

[28] J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk, “On the value of static analysis for fault detection in software,” *IEEE Trans. Software Eng.*, vol. 32, no. 4, pp. 240–253, Apr. 2006, doi: 10.1109/TSE.2006.38.

[29] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, “Using Static Analysis to Find Bugs,” *IEEE Softw.*, vol. 25, no. 5, pp. 22–29, Sep. 2008, doi: 10.1109/MS.2008.130.

[30] H. M. Kienle, J. Kraft, and T. Nolte, “System-specific static code analyses: a case study in the complex embedded systems domain,” *Software Quality Journal*, vol. 20, no. 2, pp. 337–367, Jun. 2012, doi: 10.1007/s11219-011-9138-7.

[31] D. Baca, B. Carlsson, K. Petersen, and L. Lundberg, “Improving software security with static automated code analysis in an industry setting,” *Softw. Pract. Exp.*, vol. 43, no. 3, pp. 259–279, Mar. 2013, doi: 10.1002/spe.2109.

[32] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, “Lessons from building static analysis tools at Google,” *Commun. ACM*, vol. 61, no. 4, pp. 58–66, Mar. 2018, doi: 10.1145/3188720.

[33] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, “How developers engage with static analysis tools in different contexts,” *Empirical Software Engineering*, vol. 25, no. 2, pp. 1419–1457, Mar. 2020, doi: 10.1007/s10664-019-09750-5.

[34] R. Gu *et al.*, “Towards Efficient Large-Scale Interprocedural Program Static Analysis on Distributed Data-Parallel Computation,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 4, pp. 867–883, Apr. 2021, doi: 10.1109/TPDS.2020.3036190.

[35] F. Pecorelli, D. Di Nucci, C. De Roover, and A. De Lucia, “A large empirical assessment of the role of data balancing in machine-learning-based code smell detection,” *J. Syst. Softw.*, vol. 169, p. 110693, Nov. 2020, doi: 10.1016/j.jss.2020.110693.

[36] F. Pecorelli, S. Lujan, V. Lenarduzzi, F. Palomba, and A. De Lucia, “On the adequacy of static analysis warnings with respect to code smell prediction,” *Empir. Softw. Eng.*, vol. 27, no. 3, p. 64, Mar. 2022, doi: 10.1007/s10664-022-10126-5.

[37] F. Pecorelli, F. Palomba, D. Di Nucci, and A. De Lucia, “Comparing Heuristic and Machine Learning Approaches for Metric-Based Code Smell Detection,” in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, May 2019, pp. 93–104. doi: 10.1109/ICPC.2019.00023.

[38] D. Di Nucci, F. Palomba, D. A. Tamburri, A. Serebrenik, and A. De Lucia, “Detecting code smells using machine learning techniques: Are we there yet?,” in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Mar. 2018, pp. 612–621. doi: 10.1109/SANER.2018.8330266.

[39] F. Arcelli Fontana, M. V. Mäntylä, M. Zanoni, and A. Marino, “Comparing and experimenting machine learning techniques for code smell detection,” *Empirical Software Engineering*, vol. 21, no. 3, pp. 1143–1191, Jun. 2016, doi: 10.1007/s10664-015-9378-4.

[40] S. Lujan, F. Pecorelli, F. Palomba, A. De Lucia, and V. Lenarduzzi, “A preliminary study on the adequacy of static analysis warnings with respect to code smell prediction,” presented at the ESEC/FSE ’20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual USA, Nov. 2020. doi: 10.1145/3416505.3423559.

[41] G. Catolino, F. Palomba, F. A. Fontana, A. De Lucia, A. Zaidman, and F. Ferrucci, “Improving change prediction models with code smell-related information,” *Empirical Software Engineering*, vol. 25, no. 1, pp. 49–95, Jan. 2020, doi: 10.1007/s10664-019-09739-0.

[42] F. Pecorelli, F. Palomba, F. Khomh, and A. De Lucia, “Developer-driven code smell prioritization,” presented at the MSR ’20: 17th International Conference on Mining Software Repositories, Seoul Republic of Korea, Jun. 2020. doi: 10.1145/3379597.3387457.

[43] S. Shcherban, P. Liang, A. Tahir, and X. Li, “Automatic Identification of Code Smell Discussions on Stack Overflow: A Preliminary Investigation,” in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bari, Italy, Oct. 2020, pp. 1–6. doi: 10.1145/3382494.3422161.

[44] A. K. Das, S. Yadav, and S. Dhal, “Detecting Code Smells using Deep Learning,” in *TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON)*, Oct. 2019, pp. 2081–2086. doi: 10.1109/TENCON.2019.8929628.

[45] B. H. Dang, “A Practical Approach for Ranking Software Warnings from Multiple Static Code Analysis Reports,” in *2020 SoutheastCon*, Mar. 2020, vol. 2, pp. 1–7. doi: 10.1109/SoutheastCon44009.2020.9368277.

[46] D. Wang, H. Zhang, R. Liu, M. Lin, and W. Wu, “Predicting bugs’ components via mining bug reports,” *J. Softw. Maint. Evol.: Res. Pract.*, vol. 7, no. 5, Apr. 2012, doi: 10.4304/jsw.7.5.1149-1154.

[47] T. Boland and P. E. Black, “Juliet 1.1 C/C++ and Java Test Suite,” *Computer* , vol. 45, no. 10, pp. 88–90, Oct. 2012, doi: 10.1109/MC.2012.345.

[48] S. Allier, N. Anquetil, A. Hora, and S. Ducasse, “A Framework to Compare Alert Ranking Algorithms,” in *2012 19th Working Conference on Reverse Engineering*, Oct. 2012, pp. 277–285. doi: 10.1109/WCRE.2012.37.

[49] M. Sharma, P. Bedi, K. K. Chaturvedi, and V. B. Singh, “Predicting the priority of a reported bug using machine learning techniques and cross project validation,” in *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, Nov. 2012, pp. 539–545. doi: 10.1109/ISDA.2012.6416595.

[50] H. Shen, J. Fang, and J. Zhao, “EFindBugs: Effective Error Ranking for FindBugs,” in *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*, Mar. 2011, pp. 299–308. doi: 10.1109/ICST.2011.51.

[51] G. Liang, L. Wu, Q. Wu, Q. Wang, T. Xie, and H. Mei, “Automatic construction of an effective training set for prioritizing static analysis warnings,” in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, Antwerp, Belgium, Sep. 2010, pp. 93–102. doi: 10.1145/1858996.1859013.

[52] M. G. Nanda, M. Gupta, S. Sinha, S. Chandra, D. Schmidt, and P. Balachandran, “Making defect-finding tools work for you,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, Cape Town, South Africa, May 2010, pp. 99–108. doi: 10.1145/1810295.1810310.

[53] S. Heckman and L. Williams, “A Model Building Process for Identifying Actionable Static Analysis Alerts,” in *2009 International Conference on Software Testing Verification and Validation*, Apr. 2009, pp. 161–170. doi: 10.1109/ICST.2009.45.

[54] P. Chen *et al.*, “IntFinder: Automatically detecting integer bugs in x86 binary program,” in *Information and Communications Security*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 336–345. doi: 10.1007/978-3-642-11145-7_26.

[55] D. S. Mendonça and M. Kalinowski, “An empirical investigation on the challenges of creating custom static analysis rules for defect localization,” *Software Quality Journal*, Jan. 2022, doi: 10.1007/s11219-021-09580-z.

[56] D. Serban, B. Golsteijn, R. Holdorp, and A. Serebrenik, “SAW-BOT: Proposing Fixes for Static Analysis Warnings with GitHub Suggestions,” in *2021 IEEE/ACM Third International Workshop on Bots in Software Engineering (BotSE)*, Jun. 2021, pp. 26–30. doi: 10.1109/BotSE52550.2021.00013.

[57] D. Marcilio, C. A. Furia, R. Bonifácio, and G. Pinto, “SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings,” *J. Syst. Softw.*, vol. 168, p. 110671, Oct. 2020, doi: 10.1016/j.jss.2020.110671.

[58] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, “Are SonarQube Rules Inducing Bugs?,” in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb. 2020, pp. 501–511. doi: 10.1109/SANER48275.2020.9054821.

[59] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandè, “AVATAR: Fixing

Semantic Bugs with Fix Patterns of Static Analysis Violations,” in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Feb. 2019, pp. 1–12. doi: 10.1109/SANER.2019.8667970.

[60] R. Bavishi, H. Yoshida, and M. R. Prasad, “Phoenix: automated data-driven synthesis of repairs for static analysis violations,” in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Tallinn, Estonia, Aug. 2019, pp. 613–624. doi: 10.1145/3338906.3338952.

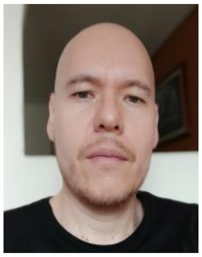
[61] M. Wyrich and J. Bogner, “Towards an Autonomous Bot for Automatic Source Code Refactoring,” in *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*, May 2019, pp. 24–28. doi: 10.1109/BotSE.2019.00015.

[62] B. Aloraini, M. Nagappan, D. M. German, S. Hayashi, and Y. Higo, “An empirical study of security warnings from static application security testing tools,” *J. Syst. Softw.*, vol. 158, p. 110427, Dec. 2019, doi: 10.1016/j.jss.2019.110427.

[63] E. A. Alikhashashneh, R. R. Raje, and J. H. Hill, “Using Machine Learning Techniques to Classify and Predict Static Code Analysis Tool Warnings,” in *2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)*, Oct. 2018, pp. 1–8. doi: 10.1109/AICCSA.2018.8612819.

Gustavo Adolfo Ruiz Montaño1. Ingeniero en Sistemas graduado de la Institución Universitaria Antonio Jose Camacho en 2017, Tecnico Profesional en Procesos Empresariales graduado de la Corporación Universitaria Centro Superior de Cali en 2009, Desarrollador de software con más de 5 años de experiencia en el desarrollo de software a la medida, experiencia certificada en diferentes clientes de reconocimiento como grupo Sura y Cencosud de Scotiabank. Actualmente me desempeño como desarrollador Senior Java para la empresa IMAGEMAKER COLOMBIA SAS, Intereses en la aplicación de las tecnologías de la información para facilitar los procesos e interés en big data y la inteligencia de negocios.
ORCID: https://orcid.org/0000-0002-3877-0284

Sebastian Robledo Giraldo. Investigador Asociado categorizado por Minciencias, Ingeniero Industrial, Magíster en Administración y Doctor en Ingeniería. Ha trabajado en las áreas de ciencias sociales e ingeniería. En ciencias sociales, he estudiado la difusión de productos a través de redes sociales sin un incentivo monetario dentro de un contexto de mercadeo emprendedor. En ingeniería, ha creado herramientas tecnológicas para el análisis de datos cienciométricos (Tree of Science). Este producto se encuentra registrado por parte de la Universidad Nacional de Colombia. También realizó una estancia postdoctoral en el Centro de Bioinformática y Biología Computacional de Colombia (BIOS) donde se realizaron desarrollos para el análisis cienciométrico. Actualmente, es docente-investigador de la Universidad Católica Luis Amigó y director de la corporación Core of Science, entidad sin ánimo de lucro que se dedica a fomentar la formación de científicos de datos.
ORCID: https://orcid.org/0000-0003-4357-4402



Huber Hernando Morales. En 2013 recibió el título de Magíster en Educación de la Universidad de Antioquia, Medellín, Colombia, en el 2009 el título de ingeniero electrónico de la Universidad de Antioquia, Medellín, Colombia. Experiencia profesional de más de 6 años como Docente universitario y 5 años como director de programas. Las principales áreas de interés en investigación son co-creación, análisis de datos, Blockchain, desarrollo de software y educación.
ORCID: https://orcid.org/0000-0003-2498-8431.