Gamifying Static Analysis

Lisa Nguyen Quang Do
Paderborn University, Germany
lisa.nguyen@upb.de

Eric Bodden
Paderborn University & Fraunhofer IEM, Germany
eric.bodden@uni-paderborn.de

ABSTRACT

In the past decades, static code analysis has become a prevalent means to detect bugs and security vulnerabilities in software systems. As software becomes more complex, analysis tools also report lists of increasingly complex warnings that developers need to address on a daily basis. The novel insight we present in this work is that static analysis tools and *video games* both require users to take on repetitive and challenging tasks. Importantly, though, while good video games manage to keep players engaged, static analysis tools are notorious for their lacking user experience, which prevents developers from using them to their full potential, frequently resulting in dissatisfaction and even tool abandonment. We show parallels between gaming and using static analysis tools, and advocate that the user-experience issues of analysis tools can be addressed by looking at the analysis tooling system as a whole, and by integrating gaming elements that keep users engaged, such as providing immediate and clear feedback, collaborative problem solving, or motivators such as points and badges.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in interaction design**; • **Theory of computation** → **Program analysis**; • **Applied computing** → *Computer games*;

KEYWORDS

Program analysis, Gamification, Integrated Environments

ACM Reference Format:
Lisa Nguyen Quang Do and Eric Bodden. 2018. Gamifying Static Analysis. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3236024.3264830

1 INTRODUCTION

To efficiently fix a warning yielded by a static analysis tool, developers have to achieve three main goals: (1) understand the code base, (2) understand the warning in the context of the code base, and (3) determine the most efficient way of fixing it. This is similar to many video games where players need (1) a good understanding of the game's universe, (2) an understanding of the quest or level they are trying to solve, and (3) to elaborate a strategy to solve it. In both cases, players and developers engage in solving quest after quest, or warning after warning over the span of hours. The difference between analysis tools and successful games is that the latter provide more support and engaging features to help the player achieve their goal (quests are well explained, incentives are made clear, hints are given, etc.), which the former typically do not offer.

Past research in static analysis has explored how to make analysis tools more usable by, for example, making them faster [20, 28, 36], classifying warning lists with more comprehensive criteria [18, 22, 31], or improving their user interfaces (UI) [11, 32]. We advocate that building a good analysis tool should include such improvements only if they are helpful *and* engaging to the developer. Such improvements also include areas that are less researched, such as encouraging collaborations [35] for example. An important point is that the analysis tool should be coherent as a system, i.e., it should be much more than using an out-of-the-box analysis algorithm and reporting its results to the developer after post-processing them. Useful features should be integrated into the design of the system, even into the analysis algorithm, if needed.

We envision a static analysis tool which not only reports bugs, but also helps developers understand the code base, and helps them fix warnings in an engaging, motivating way. In that sense, the tool is really an intelligent code assistant. To achieve this, we propose to leverage the knowledge of game designers, and to integrate gaming elements into analysis tools to improve their user experience. Static analysis is very powerful, but the tooling is unusable, it cannot be used to its full potential. With this paper, we wish to motivate the need for creating useful, complete analysis tools.

In Section 2, we explain how user-expercience issues of traditional static analysis tools can impede self-motivation. Then, we initiate a discussion on how to apply gaming elements to static analysis in Section 3. We report initial results on the acceptance of gamified static analysis features in Section 4. Section 5 details the related work and Section 6 concludes.

2 SELF-MOTIVATION IN STATIC ANALYSIS

The Self-Determination Theory (SDT) defines three innate psychological needs which influence self-motivation: competence, relatedness and autonomy [26]. In the context of video games, those needs have been found to be good predictors of enjoyment and satisfaction [27]. *Competence* refers to a need for a challenge and its subsequent sense of achievement (e.g., having learnt new skills). *Autonomy* is about the control of the player on their own actions (i.e., the degree of choice offered at each step of the game). *Relatedness* refers to the sense that the player's actions have consequences on the universe of the game.

A static analysis models all possible runtime scenarios from the source code only. Such complex operations can take a long time to complete [17]. Used as such, analyses cannot recompute updates in a short time, so until the entire analysis is re-run, developers do

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11…$15.00
https://doi.org/10.1145/3236024.3264830

Figure 1: Workflow of a traditional use of static analysis. 1–4 show where gamification can be applied.

not know if their fix worked, if it didn't, or if it introduced a new bug. This impedes *relatedness* and *competence*. Static analysis is a case where too much *autonomy* is given to the developer: to build its own model, an analysis makes assumptions that may differ from the developer's understanding of the code. This makes it harder for developers to understand why the analysis reports a warning and if it is relevant to them, leaving them on their own as to how to fix it [6, 17]. This traditional way of using static analysis can thus be detrimental to the developer's experience.

3 CHALLENGES

When thinking of games, concepts such as points, badges, or profiles immediately come to mind. However, it is important to remember that the goal of gamifying a system is not about making it a game but making the tool more engaging to the users, and also considering when *not* to gamify [5, 14]. All features of a good analysis tool should be *engaging*, *useful* (i.e., directly assist the developer in fixing bugs), and be minimally disruptive of the developer's work [12, 16, 24]. This last point is all the more important as static analysis already requires the developer to learn about the code base, warning information, and fixing techniques. Adding gaming abstractions on top of this (e.g., quests) could be distracting or confusing.

Figure 1 presents a traditional use of a static analysis tool: the analysis is run on a separate server —typically as part of nightly builds, and the warnings are reported to developers who address them on the next morning. In the following, we focus on ①–④ and detail the challenges raised by gamifying static analysis.

① **Responsiveness:** Because static analysis can take a long time to terminate, analysis tools seldom provide support for providing immediate feedback in response to a developer modification of source code. To improve this, we raise two challenges.

(a) *Making the analysis responsive.* Past approaches to make analyses faster such as incremental [28] or just-in-time analyses [20] can handle code updates quick enough to provide immediate feedback. However, worst cases can still run for longer than the original analysis would. This must be avoided when designing a gamified tool: a responsive interface cannot wait for the analysis to complete, even for a fraction of the code changes. Research in this direction needs to guarantee a maximum re-computation time of one second in *all* cases (Nielsen's threshold for interactive UIs [25]). This could be done by avoiding or pre-computing known worst cases, translating them in easily verifiable heuristics, or by reporting different types of warnings at different points of the software development lifecycle for example.

(b) *Making the UI responsive.* A key component of good games is their short response time. This is typically ensured through visual or sound effects on a game event or a player's action. Gamified tools should also follow this principle, for *relatedness*. For example, when a developer fixes a bug, warning lists should be immediately updated, and present warnings in a way that makes it easy to identify which ones are newly created or fixed, which is difficult for a large number of warnings. Gamified analysis tools need UIs that supports such mechanisms.

② **Solution-oriented communication:** Analysis tools typically display information in terms of bugs (e.g., vulnerability types, severity etc.). Instead, we propose to shift the focus towards fixes, which revolves around the following three challenges.

(a) *Presenting warnings using fix information.* In video games, players typically choose quests they can handle at their current level. In most analysis tools, fix information is rarely available and requires from the developer to first look into the bug and estimate the effort of fixing it. Better analysis tools could provide an estimate of this effort, or even propose quick fixes (incurring the danger of introducing other bugs). Displaying fixes instead of bugs can also eliminate the need to triage through lists of bugs, and give developers direct information they could act on (e.g., prioritize fixes which have the most impact). This would spare developer effort, and give them more visibility.

(b) *Learning from the developer.* In some cases, the developer knows or can calculate information the analysis is not able to compute, in particular constraints on runtime values. Tools could query users for missing information and integrate it back into the analysis. The process of guiding the developer to obtain such information based on what the analysis knows, and how to communicate with them in a human-readable, engaging way is almost like a mini-game inside of the gamified tool.

(c) *Pacing the difficulty.* The levels of games such as Tetris typically increase in difficulty. Sometimes, a particularly difficult level can appear, and solving it motivates the player to continue playing. Ordering lists of static analysis warnings by fix difficulty would help developers learn about the tool, bugs, and code base with a gentler learning curve. Another important concept is minimizing external influences. For example, Tetris tiles are always the same, and the goal and rules never change, so players can just concentrate on their task: aligning tiles. Grouping warnings with similar fixes would have a similar effect, allowing developers to concentrate on the fixing task without having to switch from one code base (or warning type) to another.

③ **Clear working status:** Video game UIs strive to give users a large *autonomy* by: (1) clearly presenting the player's current status to help them choose their next action, and (2) providing them with intuitive actionable controls to take those actions. The UI of static analysis tools should also address those challenges.

(a) *Showing the developer's current status.* Just like video games, there are two main phases to using a static analysis tool: selecting which warnings (quests) to work on, and fixing those warnings. In the first phase, developers need an overview of all warnings, of the warnings they want to fix, and clear information that helps them select warnings to fix (e.g., incentives or impact of the fix). In the second phase, developers focus on one warning in particular, so they need detailed information about that one warning. In all cases, transparency over how the

analysis works is key. For example, if the tool cannot determine if the warning has been fixed without re-running the analysis, it should make it clear in the interface, and perhaps put the warning in a waiting list for the next day.

(b) *Showing available actions.* Another important element is to always provide developers with the actions they can take (e.g., cancel a fix, assign themselves a warning, etc.) without cluttering the view with actionable items, and providing support for those actions (e.g., a quick rollback system for a cancel action). Designing such a UI is key to good usability.

④ **Teamwork:** An aspect of static analysis that is often overlooked by analysis writers is that, code can be developed collaboratively. Leveraging a group's knowledge and experience of the tool and the code base can be very efficient, and raises two challenges:

(a) *Collaborative problem-solving.* Creating a collaborative debugging environment where the strengths of different team members are capitalized on without working against each other can be tricky. For example, suggesting colleagues that might have fixed similar issues without flooding said colleagues with too many requests. Whether interactions between developers are limited to questions and answers or the tool integrates real-time interactions (similar to Google Docs), it is important for a gamified tool to create an environment where the developer can ask for help without being penalized, or leave the warning for later, when they have more experience.

(b) *Motivating elements for collaborative work.* With the collaboration of different individuals comes incentives and reward systems such as points, badges, levels, achievements, rankings, etc. Such elements can also be applied to a single user, to help them have a clearer overview of what they are doing on a particular day. Such a system could be counter-productive since it sets goals (e.g., earning points) that are different from the original goal of fixing bugs. Researching which elements to use and how to carefully balance them is a challenging task.

4 EARLY RESULTS

We have built an initial UI prototype of a gamified static analysis tool addressing the challenges from Section 3. Figures 2 and 3 show the selection screen and a close-up of the debug screen mentioned in ③(a). The tool features Ⓐ–Ⓠ are described in Table 1. While designing the features, we have focused on two aspects: customizing the information based on the developer's current work (e.g., their currently assigned bugs) and experience (Ⓗ and Ⓟ), and providing them with as focused feedback and actionable actions (e.g., Ⓚ presents information embedded in the code where it is relevant: explanations on why the tool reports a bug, its relationship to other bugs, and access to possible actions: "This is wrong", "I fixed it", etc. Ⓠ appears when the developer fixes a bug. It shows them their status (new points, achievements...) and gives them access to possible actions: "Get more bugs to fix").

We ran a 45-minutes cognitive walkthrough of the prototype with eight researchers who have knowledge of how static analysis tools function. Five of them had worked with analysis tools as developers in the past. Participants performed 23 tasks grouped in five themes: navigate the selection screen, (un-)assign bugs, navigate the debug screen, fix a bug, mark a bug as a false positive. In an interview, we asked participants whether they found the tool features (1) useful to complete their tasks and (2) engaging (i.e., they enjoyed using it). Finally, we asked them to list the top features of the tool. The study protocol and results are made available online [1].

Table 1: List of the gamified features in Figures 2–3, the percentage of participants having found them useful (U) / engaging (E) to achieve their tasks, and having mentionned them among their preferred ones (P).

	Feature description	%U	%E	%P
Ⓐ	Developer profile.	50	75	25
Ⓑ	Point system.	12.5	37.5	25
Ⓒ	Badges.	37.5	62.5	12.5
Ⓓ	Overview of yesterday's achievements.	62.5	75	37.5
Ⓔ	Map/overview of the warnings.	75	37.5	0
Ⓕ	Filtering functionalities for the map.	100	50	50
Ⓖ	Warnings assigned to the user.	100	33.3	25
Ⓗ	Suggestions of bugs to fix.	100	16.7	0
Ⓘ	Assignment system.	100	25	0
Ⓙ	Warning history.	62.5	37.5	0
Ⓚ	Warning information in the code.	100	62.5	87.5
Ⓛ	Gutter icons.	85.7	0	12.5
Ⓜ	Mark false positives ("This is wrong").	100	62.5	12.5
Ⓝ	Mark as fixed ("I fixed it").	100	80	12.5
Ⓞ	Cancel Ⓜ or Ⓝ with one click.	100	0	0
Ⓟ	Fix suggestions.	100	50	50
Ⓠ	Notification popup.	87.5	87.5	37.5

Table 1 presents the results of the interview. We see that features Ⓓ–Ⓟ were perceived as useful by a strong majority of the participants. In particular, the information embedded in the code Ⓚ was mentioned by 87.5% of the participants as part of their top useful features, confirming the need for answers to challenges ③(a) and ③(b). Next come the filtering functionalities Ⓕ (50%, ③(a) and ①(b)) and fix suggestions Ⓟ (50%, ②(a)).

Features Ⓐ–Ⓒ, which correspond to typical gaming features (badges, profile, points), have a lower usefulness score, and have been perceived as more engaging than useful by the participants. Many participants overlooked them, as they "are unrelated to what I am doing". In contrast, feature Ⓠ (notifications) is also a typical gaming feature, but received a much higher usefulness and engagement score (87.5%), and was also mentioned as one of the top functionalities by 37.2% of the participants. This feature matches challenges ①(b), ③(a), and ③(b). This suggests that for static analysis, useful gamification features should remain as discrete as possible, and only be used when they provide useful information, which confirms observations made by other researchers on the more general field of gamifying software engineering [24].

5 RELATED WORK

In this section, we present related work in the areas of gamification and of the challenges from Section 3. Because of space limitations, we will only discuss their applications to static analysis.

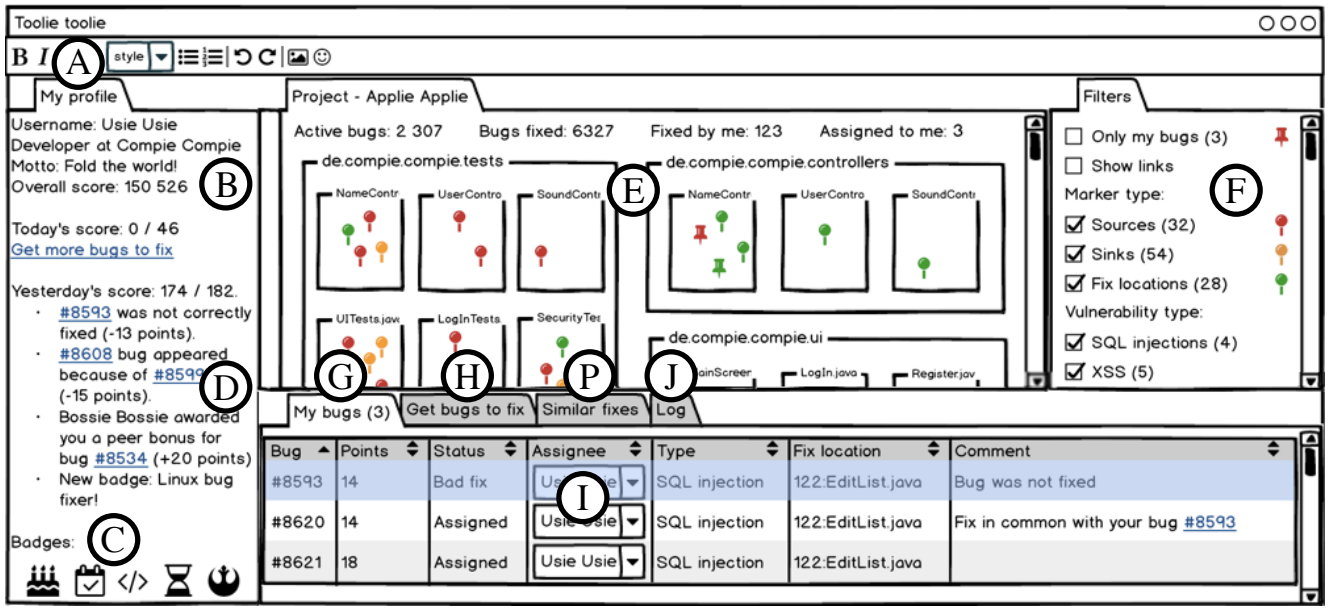

Figure 2: Selection screen. A–J and P are detailed in Table 1 and Section 4.

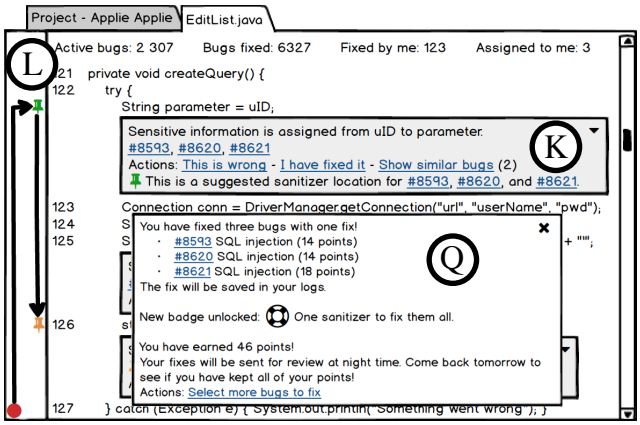

Figure 3: Close-up of the debug screen. K–Q are detailed in Table 1 and Section 4. M–O are included in the grey box (K).

Gamification: Many approaches to gamifying software engineering have been researched in the past decade [5, 10, 13, 16, 34], mostly using points and badges. Recent literature proposes frameworks for gamification of general software systems [3, 24, 30]. In this paper, we motivate the need for a more concrete approach of gamifying static analysis tools. To our best knowledge, the only such attempt was limited to assigning points to the warnings [2]. We advocate for a more complete gamification of the entire analysis system.

Responsiveness of static analysis: Past and current approaches strive to make static analysis faster. Incremental analysis re-runs only on incremental change sets [28, 36]. The just-in-time approach prioritizes certain analysis directions [20]. Frameworks such as Tricorder [29] or Parfait [9] run quick and imprecise analyses first, and then refine the results with longer-running analyses.

Solution-oriented static analysis: Many analysis tools run post-processing modules that compute hints to guide the developer fix warnings. This ranges from showing pages of the vulnerability description [23] to computing vulnerability graphs [7, 21] to querying developers for generating quick fixes [4].

Usability of static analysis: Usability issues in static analysis tools have been documented over decades of use [6, 8, 17, 19, 33]. Approaches for improving particular UI components are explored in academia and industry: navigating program flows [32], integration of analysis tools in the workflow [8], graph visualisations [15], etc.

Collaboration in static analysis: From real-time collaboration (Google Docs), to management software (GitHub) to crowdsourcing [35], using knowledge from multiple individuals has brought better user experience. To our best knowledge, this has not yet been researched for static analysis.

6 CONCLUSION

We have presented the novel idea of following gaming principles to improve the usability and engagement capabilities of static analysis tools. We propose going beyond simply including points and badges, and look at the analysis system as a whole to define gamified features. Our preliminary study shows that such features are well-received, and motivates the need for creating developer-centric static analysis tools. Building such a system may require changes to the way current analysis tools and analysis algorithms typically work, by making them responsive, computing fix-centered warnings instead of bug-centered ones, improving the UI of the tools and adding collaborative features, challenges that are —for some— not been widely researched in the context of static analysis.

ACKNOWLEDGMENTS

This research has been funded by the Heinz Nixdorf Foundation.

REFERENCES

[1] 2018. Cognitive walkthrough artifacts. https://blogs.uni-paderborn.de/sse/tools/gamifying-static-analysis.

[2] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa. 2014. A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools. In *2014 6th International Workshop on Empirical Software Engineering in Practice*. 37–42. https://doi.org/10.1109/IWESEP.2014.17

[3] T. Barik, E. Murphy-Hill, and T. Zimmermann. 2016. A perspective on blending programming environments and games: Beyond points, badges, and leaderboards. In *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 134–142. https://doi.org/10.1109/VLHCC.2016.7739676

[4] T. Barik, Y. Song, B. Johnson, and E. Murphy-Hill. 2016. From Quick Fixes to Slow Fixes: Reimagining Static Analysis Resolutions to Enable Design Space Exploration. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 211–221. https://doi.org/10.1109/ICSME.2016.63

[5] K. Berkling and C. Thomas. 2013. Gamification of a Software Engineering course and a detailed analysis of the factors that lead to it's failure. In *2013 International Conference on Interactive Collaborative Learning (ICL)*. 525–530. https://doi.org/10.1109/ICL.2013.6644642

[6] Al Bessey, Ken Block, Benjamin Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles-Henri Gros, Asya Kamsky, Scott McPeak, and Dawson R. Engler. 2010. A few billion lines of code later: using static analysis to find bugs in the real world. *Communications of the ACM* 53, 2 (2010), 66–75. https://doi.org/10.1145/1646353.1646374

[7] Checkmarx. 2018. Checkmarx home page. https://www.checkmarx.com/.

[8] Maria Christakis and Christian Bird. 2016. What developers want and need from program analysis: an empirical study. In *International Conference on Automated Software Engineering (ASE)*. 332–343.

[9] Cristina Cifuentes, Nathan Keynes, Lian Li, Nathan Hawes, and Manuel Valdiviezo. 2012. Transitioning Parfait into a Development Tool. *IEEE Security & Privacy* 10, 3 (2012), 16–23. https://doi.org/10.1109/MSP.2012.30

[10] M. R. d. A. Souza, K. F. Constantino, L. F. Veado, and E. M. L. Figueiredo. 2017. Gamification in Software Engineering Education: An Empirical Study. In *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)*. 276–284. https://doi.org/10.1109/CSEET.2017.51

[11] L. N. Q. Do, K. Ali, B. Livshits, E. Bodden, J. Smith, and E. Murphy-Hill. 2017. Cheetah: just-in-time taint analysis for android apps. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 39–42. https://doi.org/10.1109/ICSE-C.2017.20

[12] Matthieu Foucault, Xavier Blanc, Margaret-Anne D. Storey, Jean-Rémy Falleri, and Cédric Teyton. 2018. Gamification: a Game Changer for Managing Technical Debt? A Design Study. *CoRR* abs/1802.02693 (2018). arXiv:1802.02693 http://arxiv.org/abs/1802.02693

[13] Flix Garca, Oscar Pedreira, Mario Piattini, Ana Cerdeira-Pena, and Miguel Penabad. 2017. A Framework for Gamification in Software Engineering. *J. Syst. Softw.* 132, C (Oct. 2017), 21–40. https://doi.org/10.1016/j.jss.2017.06.021

[14] Gartner. 2018. Gartner Says by 2014, 80 Percent of Current Gamified Applications Will Fail to Meet Business Objectives Primarily Due to Poor Design. https://www.gartner.com/newsroom/id/2251015.

[15] Grammatech. 2018. CodeSonar home page. https://www.grammatech.com/products/codesonar.

[16] Scott Grant and Buddy Betts. 2013. Encouraging User Behaviour with Achievements: An Empirical Study. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. IEEE Press, Piscataway, NJ, USA, 65–68. http://dl.acm.org/citation.cfm?id=2487085.2487101

[17] Brittany Johnson, Yoonki Song, Emerson R. Murphy-Hill, and Robert W. Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *International Conference on Software Engineering (ICSE)*. 672–681. http://dl.acm.org/citation.cfm?id=2486877

[18] Woosuk Lee, Wonchan Lee, Dongok Kang, Kihong Heo, Hakjoo Oh, and Kwangkeun Yi. 2017. Sound Non-Statistical Clustering of Static Analysis Alarms. *ACM Trans. Program. Lang. Syst.* 39, 4, Article 16 (Aug. 2017), 35 pages. https://doi.org/10.1145/3095021

[19] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E. James Whitehead Jr. 2013. Does bug prediction support human developers? Findings from a Google case study. In *International Conference on Software Engineering (ICSE)*. 372–381. http://dl.acm.org/citation.cfm?id=2486838

[20] Nguyen Quang Do Lisa, Ali Karim, Livshits Benjamin, Bodden Eric, Smith Justin, and Murphy-Hill Emerson. 2017. Just-in-time Static Analysis. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2017)*. ACM, New York, NY, USA, 307–317. https://doi.org/10.1145/3092703.3092705

[21] Benjamin Livshits and Stephen Chong. 2013. Towards Fully Automatic Placement of Security Sanitizers and Declassifiers. *SIGPLAN Not.* 48, 1 (Jan. 2013), 385–398. https://doi.org/10.1145/2480359.2429115

[22] Ravi Mangal, Xin Zhang, Aditya V. Nori, and Mayur Naik. 2015. A User-guided Approach to Program Analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 462–473. https://doi.org/10.1145/2786805.2786851

[23] MITRE. 2018. Common Weakness Enumeration. https://cwe.mitre.org/.

[24] Scott Nicholson. 2012. A User-Centered Theoretical Framework for Meaningful Gamification. In *Games+Learning+Society 8.0*.

[25] Jakob Nielsen. 1994. *Usability Engineering*. Elsevier.

[26] Richard M. Ryan and Edward L. Deci. 2000. Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being. *American Psychologist* (2000). https://doi.org/10.1037/0003-066X.55.1.68

[27] Richard M. Ryan, C. Scott Rigby, and Andrew Przybylski. 2006. The Motivational Pull of Video Games: A Self-Determination Theory Approach. *Motivation and Emotion* 40, 4 (2006), 344–360. https://doi.org/0.1007/s11031-006-9051-8

[28] Barbara G. Ryder. 1983. Incremental Data Flow Analysis. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL '83)*. ACM, New York, NY, USA, 167–176. https://doi.org/10.1145/567067.567084

[29] Caitlin Sadowski, Jeffrey Van Gogh, Ciera Jaspan, Emma Söderberg, and Collin Winter. 2015. Tricorder: Building a Program Analysis Ecosystem. In *International Conference on Software Engineering (ICSE)*. 598–608.

[30] T. Dal Sasso, A. Mocci, M. Lanza, and E. Mastrodicasa. 2017. How to gamify software engineering. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 261–271. https://doi.org/10.1109/SANER.2017.7884627

[31] Mark S. Sherriff, Sarah Smith Heckman, J. Michael Lake, and Laurie A. Williams. 2007. Using Groupings of Static Analysis Alerts to Identify Files Likely to Contain Field Failures. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC-FSE '07)*. ACM, New York, NY, USA, 565–568. https://doi.org/10.1145/1287624.1287711

[32] J. Smith, C. Brown, and E. Murphy-Hill. 2017. Flower: Navigating program flow in the IDE. In *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 19–23. https://doi.org/10.1109/VLHCC.2017.8103445

[33] Justin Smith, Brittany Johnson, Emerson R. Murphy-Hill, Bill Chu, and Heather Richter Lipford. 2015. Questions developers ask while diagnosing potential security vulnerabilities with static analysis. In *Foundations of Software Engineering (FSE)*. 248–259.

[34] Nikolai Tillmann, Jonathan De Halleux, Tao Xie, Sumit Gulwani, and Judith Bishop. 2013. Teaching and Learning Programming and Software Engineering via Interactive Gaming. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 1117–1126. http://dl.acm.org/citation.cfm?id=2486788.2486941

[35] M. C. Yuen, I. King, and K. S. Leung. 2011. A Survey of Crowdsourcing Systems. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 766–773. https://doi.org/10.1109/PASSAT/SocialCom.2011.203

[36] Sheng Zhan and Jeff Huang. 2016. ECHO: Instantaneous in Situ Race Detection in the IDE. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, New York, NY, USA, 775–786. https://doi.org/10.1145/2950290.2950332