Summarization Techniques for Code, Change, Testing, and User Feedback (Invited Paper)

Sebastiano Panichella
University of Zurich

***Abstract*—Most of today's industries, from engineering to agriculture to health, are run on software. In such a context, ensuring software quality play an important role in most of current working environment and have a direct impact in any scientific and technical discipline. Software maintenance and testing have the crucial goal to find or discover possible software bugs (or defects) as early as possible, enabling software quality assurance. However, software maintenance and testing are very expensive and time-consuming activities for developers. For this reason, in the last years, several researchers in the field of Software Engineering (SE) devoted their effort in conceiving tools for boosting developers productivity during such development, maintenance and testing tasks. In this talk, I will first discuss some empirical work we performed to understand the main socio-technical challenges developers face when joining a new software project. I will discuss how to address them with the use of appropriate recommender systems aimed at supporting developers during program comprehension and maintenance tasks. Then, I'll show how Summarization Techniques are an ideal technology for supporting developers when performing testing and debugging activities. Finally, I will summarize the main research advances, the current open challenges/problems and possible future directions to exploit for boosting developers productivity.**

I. Introduction

Nowadays, most industries are running on enterprise software applications [1] and, only observing the year 2016, software bugs impacted 4.4 billion people and $1.1 trillion in assets [2]. Besides that, recent studies have also shown that improving software development, maintenance and testing processes can save at least 50% of software costs [3]. To cope with the need for more efficient software development practices [4], [5], the conventional software release cycle has evolved in recent years into a complex process [6], integrating Continuous Delivery (CD) and Continuous Integration (CI). Continuous Delivery is one of the most emerging software development practices, in which developers' code changes are sent to server machines to automate all software integration (e.g., building and testing) tasks required for the delivery [7]. When this automated process fails (known as *build failure*), developers are forced to go back to coding to discover and fix the root cause of the failure [8]–[10]; otherwise, the changes are released to production, in short cycles.

The objective of the paper is to provide a summary of the keynote of the *2nd Workshop on Validation, Analysis and Evolution of Software Tests (VST 2018)*.

Paper structure. In Section II we will discuss the main context of the talk with particular focus to the emerging needs of contemporary software development, maintenance, and testing practices. Then, Section III will describe the main topics of the talk, with particular focus to the recent adoption of *Summarization Techniques for Code, Change, Testing and User Feedback*. We will discuss the benefits of applying such techniques for supporting development, maintenance and testing tasks, highlighting potential future challenges and future directions for boosting developers productivity.

II. Overview and Motivation

A. Context and Needs

Several software companies around the world have started to adopt continuous delivery and continuous integration practices, with the benefit that high quality products can be build through rapid iterations [7], [11], [12]. However, in this context both inexperienced [13] and experienced [14], [15] developers need to deal with a huge set of necessary technologies (e.g., versioning and issue tracking systems) and software data (e.g., build logs, source and test code) [16], [17] to actively contribute to a software project [18]. This highlights the ***need for strategies to reduce the developer's cognitive load due to the large amount of information to analyze and understand while performing release cycle tasks*** [19]. Recent work proposed approaches to help developers browse heterogeneous data during maintenance tasks [17], [20], [21]. However, these approaches represent only preliminarily attempts to support developers in handling current release cycle tasks.

Program comprehension represents a vital software engineering activity which facilitates reuse, maintenance, reverse engineering and reengineering of software systems. *Maintenance* and *testing* are crucial activities for evolving software applications, to ensure software quality (e.g., by finding defects introduced during the development phases) and meeting market requirements. Differently from other activities, software maintenance and testing have the crucial goal to find or discover software bugs (or defects) as early as possible. Nowadays, software maintenance and testing are very expensive and time-consuming activities for developers. For this reason, in the last years, several researchers in the field of Software Engineering (SE) devoted their effort in conceiving tools with the goal of boosting developer productivity during development, maintenance and testing tasks. Nevertheless, ***more research is needed to design tools/prototypes to increase developer productivity while performing maintenance and testing tasks*** [22].

An essential ingredient to address the challenges of development, maintenance and testing tasks in modern release cycles

is to automatically analyze different types of data and extract relevant content to support maintenance and testing activities. Therefore, *summarization techniques* are suitable technologies to reach this goal. By definition, *'Summarization approaches'* [23] have the general capability to automatically ***extract*** or ***abstract*** key content from various sources of information, thus, condensing it. In the last years software engineering research has yield several novel techniques mostly based on *Code and Code Change Summarization Techniques* [24], [25] for supporting developers during program comprehension, maintenance and testing activities. Specifically, Murphy's dissertation [26] is the earliest work which proposed to generate code summaries by analysing structural information of the source code for supporting developer during program comprehension tasks. More recently, Sridhara *et al.* [27] suggested to use pre-defined *templates* of natural language sentences, filled with linguistic elements (verbs, nouns, etc.) extracted from important method signatures [28], for improving developers program comprehension.

I recently explored the use of summarization techniques [29] by proposing TESTDESCRIBER (TD) [30], a promising solution that combines summarization approaches with code coverage information. Empirical results showed TestDescriber ability to dramatically reduce the time developers spend in fixing bugs, increasing readability and comprehensibility of the tests. TestDescriber achieves these results by automatically generating Test Case Summaries (TCS), i.e., natural language descriptions of the portion of code exercised by each test, that help developers interpreting source/test code information, without looking through the entire code. TestDescriber generates summaries applying a template-based strategy [31], [32]. As shown in Figure 1, a set of pre-defined templates of natural language sentences is used to generate four types of summaries at four different levels of abstraction: (i) class level summaries, (ii) test method level summaries, (iii) fine-grained statement summaries, and (iv) branch coverage summaries.

We recently used summarization approaches for mining mobile apps data and cluster [33], summarize [34] or visualize [35], [36] user feedback contained in user reviews. The generated summaries help developers plan future change tasks, accommodate actual user needs and integrate them in the testing process of mobile apps [37].

The aforementioned work [29] highlights how summarization techniques can be potentially used to address several of the research challenges of contemporary SE community.

B. General Learning Goals of the Talk

In this talk we shed the light on the summarization techniques adopted in SE (i) for *summarizing software artifacts* and supporting developers during program comprehension activities; (ii) for *automatically mining requirements from user reviews* [38] of mobile applications; (iii) for supporting developers in performing specific maintenance tasks and that are based on *code change summarization techniques*; (iv) for *supporting developers during testing* and bug fixing tasks. Finally, we will discuss some challenges and opportunities in adopting such techniques in SE and its potential for the future of SE research.

```
1  /**
2   * The main class under test is TemperatureSaver.
3   *  It describes a single temperature saver and maintains information
4  regarding:
5   *  - the far of the temperature saver;
6   *  - the lower of the temperature saver;
7   * */
8  public class TemperatureSaverTest {
9  /**OVERVIEW: The test case "testTemperatureSaverDouble" covers around
10 *33.333336% [13,17,18,20,23,29] of statements.
11 *testTemperatureSaver covers 33% of this test. Same lines [23,29]
12 *testTemperatureSaverConversion covers 67% of this test. Same lines
13 [13, 17, 18, 20]*/
14 @Test
15 public void testTemperatureSaverDouble() {
16 //The test case instantiates a "TemperatureSaver" with  celsius
17 //temperature equal to 28.0, and  fahrenheit temperature equal to 0.0.
18 //The execution of this constructor implicitly covers the following 1
19 //conditions:
20 // - the condition  "celsius temperature equals to 0.0" is FALSE;
21 TemperatureSaver con1 = new  TemperatureSaver(28.0, 0.0);
22 //Then, it tests:
23 //1) whether the fahrenheit of con1 is equal to 82.4 with delta equal
24 //to 1.0; --> java.lang.AssertionError: expected:<82.4> but was:<0.0>
25 assertEquals(82.4, con1.getFahrenheit(), 1.0);
26 //2) whether the celsius of con1 is equal to 0.0 with delta equal to
27 //0.0;
28 assertEquals(0.0, con1.getCelsius(), 0.0);
29     }
30 }
```

Fig. 1. Example of summary generated by TestDescriber for the JUnit test class corresponding to the class *TemperatureSaver.java*

C. Importance for the SE Community

Most of software engineering tasks require to developers manually analyzing or inspecting different kinds of software artifacts and to understand specific parts of the source code. We argue that summarization techniques have the potential to play in the near future of SE research an important role in automating most of these tasks. Surveying the SE literature over the past 4 years we found that both traditional and less traditional SE tasks are being addressed through the use of summarization techniques and this trend increased in SE over the last few years. Some examples of such SE tasks include change impact analysis, refactoring, bug prediction, software re-documentation, bug triaging, testing of software artifacts, test suites redocumentation, mining requirements from user reviews, etc. Results of our survey highlight the high usefulness, applicability and generality of such techniques in supporting many SE tasks and covering a wide range of SE research topics. As concrete examples successful techniques have been proposed for supporting developers during (i) program comprehension, re-documentation of Java methods [24], [31] and Java classes [39]; in automatically (ii) generating high quality commit messages [40] or release notes [41]; in (iii) supporting developers (or testers) fixing more bugs during testing and/or debugging activities [30], [42].

However, beside the undisputed advantages introduced by the adoption of such proposed techniques, SE students in general and SE practitioners in particular, are not aware of the potential of summarization techniques in automating specific, difficult, time consuming and error prone tasks performed by developers during development, maintenance and testing activities. We believe that presenting this keynote at VST 2018 will allow

more SE researchers, practitioners and students to be aware about the profitable adoption of summarization techniques for SE research or for performing specific SE tasks. We believe that this would help to fill a gap in the current background of younger researchers allowing them to immediately use such techniques for producing new research in SE. Moreover, it has the potential for practitioners to have to their disposal scalable and practical solutions for speed up several of the development, maintenance and testing activities they perform on a daily basis.

D. Target Audience

The talk is planned for being suitable for both researchers, practitioners and educators and no prior knowledge is required for the understanding of the topics discussed. If needed, the audience will be referred to further readings, recommended for those who want to acquire in depth details about the discussed techniques and the methodology adopted for their validation.

III. Outline

In this talk, I will first discuss some empirical work we performed to understand the main socio-technical challenges developers face when joining a new software project. I will discuss how to address them with the use of appropriate recommender systems aimed at supporting developers during program comprehension and maintenance tasks. After that we will introduce the basics techniques adopted for the summarization of structured and unstructured software artifacts describing its adoption for particular SE tasks. Then, I'll show how **Summarization Techniques** are an ideal technology for supporting developers when performing testing and debugging activities. Finally, I will summarize the main research advances, the current open challenges/problems and possible future directions to exploit for boosting developers productivity. The detailed outline of the talk is described in the following sections.

A. Supporting Newcomers in Software Development Projects

The recent and fast expansion of OSS (Open-source software) and industrial organizations has fostered research investigating the role played by new developers in the longevity and the success of software projects. Recent work has shown that an high percentage of newcomers tend to leave software project because of the presence and persistence of *socio-technical barriers* characterizing the software organizations they are joining [18]. This is especially true in OSS organizations. In this talk we discuss the potential problems arising newcomers joining software projects, and possible solutions to support them. Among various recommenders proposed in the literature, we discuss tools that (i) suggest appropriate mentors or experts to newcomers during the training stage [43]–[47]; that (ii) support newcomers during program comprehension and maintenance tasks [48] by generating high quality source code summaries [49], [50], providing accurate descriptions of source code elements [51]–[53].

B. Summarizing Structured and Unstructured Data

This part of the talk will introduce the basic concepts behind techniques for Code Summarization, Code Change Summarization, Test Cases Summarization and App Reviews Summarization. It will continue in the next parts by talking more in details about each of the mentioned techniques presenting some related applications in the field and by discussing challenges and opportunities in adopting them in SE.

C. Source Code Summaries and Code Change Summarization

This part of the technical briefing will present high-level concepts related to source code summarization techniques and practical hints on the use of the SWUM technology [54]. Then, it will discuss how they can be leveraged for supporting developers during program comprehension [28], [50], [55], re-documentation of software artifacts [56], [57] and maintenance tasks [58], [59].

D. Summarization Techniques in the Mobile & Testing Contexts

This part of the talk will present and motivate summarization techniques in the Mobile & Testing Contexts.

In the last couple of years, we focused our attention on defining techniques and tools to mine mobile app data available in app stores [60] to support software engineers performing maintenance and evolution activities. By mining mobile app data, we proposed approaches to classify [61], [62], cluster [33], summarize [34] or visualize [35], [36] user feedback contained in user reviews and browsable in mobile app stores (e.g., Google Play Store). The resulting tools help developers plan future change tasks (by better wiring up feature development and bug fixing) and accommodate actual user needs (by means of multi-source interlinking of user requests and actual changes).

In a recent work, we proposed the idea of *user-oriented testing*, where user feedback (in form of user review in app stores) can be systematically integrated into the testing process of mobile apps [37]. In this work, we show that user feedback information can be exploited to provide contextual details about errors or exceptions detected by automated testing tools and can help detecting bugs that such tools are not able to detect. In addition, this part of the talk will concern high-level concepts related to test case summarization techniques, discussing how they can be leveraged for supporting developers (or testers) fixing more bugs during testing and/or debugging activities [30], [42], to integrate user feedback into testing of mobile applications [37] or to support CD/CI tasks [63].

E. Keynote Summary and Conclusion

Recent research in SE observed an increasing adoption of summarization techniques for accomplishing simple or more complex, development, maintenance and testing tasks. However, their adoption in industrial contexts requires substantial novel and advanced research to make them applicable in any industrial or open source organizations. Thus, future research in the field

is devoted to fill the existing gap between industrial needs and current provided research prototypes:

- *Summarization of Hetherogeneous Data*: current summarization approaches are mostly conceived for analyzing one or two sources of information. However, when performing development, maintenance and testing tasks developers access to various types of heterogenous data. Thus, future Summarization techniques should be designed with advanced mechanisms able to distill, in a simultaneous manner, the relevant knowledge present in different sources of information, presenting it in a unified manner, depending on the specific task the developers is performing.
- *Scalability and Integration in the CD/CI Process*: current summarization approaches are able to proficiently distill relevant information from various kind of software artifacts. However, they are usually computationally expensive and thus, not applicable in real working contexts. Moreover, most of such tools are difficult to integrate in the current continuous delivery software development process. Hence, future research should be devoted on designing tools able to analyze, with substantial low computational cost, the huge amount of available heterogeneous data, integrating the summarized information in the the various development phases composing the CD pipeline applied in a software organization.
- *Visualization of Software Summaries*: most of generated software summaries are presented as set of textual fragments that share similar concepts. Thus, part of the future research related to the application of source code and code change summarization, needs to be devoted to the definition of proper visualization metaphors, that actually present the information provided by the generated summaries in a more structured manner.

All materials covered in this talk will be made available online at: http://www.ifi.uzh.ch/en/seal/people/panichella.html

ACKNOWLEDGMENT

I would like to thank Cyrille Artho and Ramler Rudolf for considering me as keynote speaker of VST 2018. We also acknowledge the Swiss National Science Foundation's support for the project SURF-MobileAppsData (SNF Project No. 200021-166275).

REFERENCES

[1] M. Andreessen, "Why software is eating the world." *http://www.wsj.com/articles/SB10001424053111903480904576512250915629460*, 2016.

[2] "Software fail watch: 2016 in review - https://www.tricentis.com/resource-assets/software-fail-watch-2016/," 2016.

[3] NIST, "goo.gl/qsynhh," 2013.

[4] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," in *ESEC/FSE 2017*, 2017, pp. 197–207. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106270

[5] P. M. Duvall, "Continuous integration. patterns and antipatterns," *DZone refcard #84*, 2010. [Online]. Available: http://bit.ly/l8rfVS

[6] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for agile software development," 2001.

[7] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation*, 1st ed. Addison-Wesley Professional, 2010.

[8] M. R. Islam and M. F. Zibran, "Insights into continuous integration build failures," in *MSR 2017*, 2017, pp. 467–470. [Online]. Available: https://doi.org/10.1109/MSR.2017.30

[9] C. Ziftci and J. Reardon, "Who broke the build? Automatically identifying changes that induce test failures in continuous integration at google scale," in *ICSE-SEIP 2017*, 2017, pp. 113–122. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2017.13

[10] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. D. Penta, and S. Panichella, "A tale of CI build failures: An open source and a financial organization perspective," in *ICSME 2017*, 2017, pp. 183–193.

[11] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[12] T. Savor, M. Douglas, M. Gentili, L. Williams, K. Beck, and M. Stumm, "Continuous deployment at facebook and OANDA," in *ICSE Companion*, 2016, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/2889160.2889223

[13] S. Panichella, "Supporting newcomers in software development projects," in *ICSME 2015*, 2015, pp. 586–589. [Online]. Available: https://doi.org/10.1109/ICSM.2015.7332519

[14] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. de Vries, "Moving into a new software project landscape," in *ICSE 2010*, 2010, pp. 275–284. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806842

[15] M. Zhou and A. Mockus, "What make long term contributors: Willingness and opportunity in oss community," in *ICSE 2012*, 2012, pp. 518–528.

[16] N. Nazar, Y. Hu, and H. Jiang, "Summarizing software artifacts: A literature review," *Journal of Computer Science and Technology*, vol. 31, no. 5, pp. 883–909, Sep 2016. [Online]. Available: https://doi.org/10.1007/s11390-016-1671-1

[17] L. Ponzanelli, A. Mocci, and M. Lanza, "Summarizing complex development artifacts by mining heterogeneous data," ser. MSR '15. IEEE Press, 2015, pp. 401–405. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820518.2820571

[18] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles, "A systematic literature review on the barriers faced by newcomers to open source software projects," *Information & Software Technology*, vol. 59, pp. 67–85, 2015. [Online]. Available: https://doi.org/10.1016/j.infsof.2014.11.001

[19] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz, "The work life of developers: Activities, switches and perceived productivity," *IEEE Trans. Software Eng.*, vol. 43, no. 12, pp. 1178–1193, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2017.2656886

[20] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework," *Empirical Software Engineering*, vol. 22, no. 3, pp. 967–995, 2017. [Online]. Available: https://doi.org/10.1007/s10664-016-9457-1

[21] L. Ponzanelli, S. Scalabrino, G. Bavota, A. Mocci, R. Oliveto, M. D. Penta, and M. Lanza, "Supporting software developers with a holistic recommender system," in *ICSE 2017*, 2017, pp. 94–105. [Online]. Available: https://doi.org/10.1109/ICSE.2017.17

[22] M. Shahin, M. A. Babar, and L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices," *IEEE Access*, vol. 5, pp. 3909–3943, 2017. [Online]. Available: https://doi.org/10.1109/ACCESS.2017.2685629

[23] U. Hahn and I. Mani, "The challenges of automatic summarization," *Computer*, vol. 33, no. 11, pp. 29–36, Nov. 2000. [Online]. Available: http://dx.doi.org/10.1109/2.881692

[24] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *WCRE 2010*, pp. 35–44. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2010.13

[25] B. Fluri, M. Würsch, M. Pinzger, and H. C. Gall, "Change distilling: Tree differencing for fine-grained source code change extraction," *IEEE Trans. Software Eng.*, vol. 33, no. 11, pp. 725–743, 2007. [Online]. Available: http://dx.doi.org/10.1109/TSE.2007.70731

[26] G. C. Murphy, "Lightweight structural summarization as an aid to software evolution," Ph.D. dissertation, 1996, aAI9704521.

[27] G. Sridhara, "Automatic generation of descriptive summary comments for methods in object-oriented programs," Ph.D. dissertation, Newark, DE, USA, 2012, aAI3499878.

[28] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: an empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383–1420, 2014. [Online]. Available: https://doi.org/10.1007/s10664-013-9285-5

[29] S. Panichella, "Summarization techniques for code, change, testing and user feedback." Presented at the Vienna Software Seminar 2017, Vienna, Austria, 2017. [Online]. Available: https://www.slideshare.net/sebastianopanichella/summarization-techniques-for-code-change-testing-and-user-feedback-vss-2017

[30] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. Gall, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *ICSE 2016*. IEEE.

[31] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *ASE 2010*. ACM, pp. 43–52.

[32] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *ICPC 2014*. ACM, pp. 279–290.

[33] F. Palomba, P. Salza, A. Ciurumelea, S. Panichella, H. C. Gall, F. Ferrucci, and A. D. Lucia, "Recommending and localizing change requests for mobile apps based on user reviews," in *ICSE 2017*, pp. 106–117. [Online]. Available: https://doi.org/10.1109/ICSE.2017.18

[34] A. Di Sorbo, S. Panichella, C. V. Alexandru, J. Shimagaki, C. A. Visaggio, G. Canfora, and H. C. Gall, "What would users change in My app? Summarizing app reviews for recommending software changes," in *FSE 2016*. ACM, pp. 499–510. [Online]. Available: http://doi.acm.org/10.1145/2950290.2950299

[35] A. D. Sorbo, S. Panichella, C. V. Alexandru, C. A. Visaggio, and G. Canfora, "SURF: summarizer of user reviews feedback," in *ICSE 2017 - Companion Volume*, pp. 55–58. [Online]. Available: https://doi.org/10.1109/ICSE-C.2017.5

[36] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "ARDOC: App Reviews Development Oriented Classifier," in *Foundations of Software Engineering (FSE), 2016 ACM SIGSOFT International Symposium on the*, 2016, pp. 1023–1027.

[37] G. Grano, A. Ciurumelea, S. Panichella, F. Palomba, and H. Gall, "Exploring the integration of user feedback in automated testing of android applications," in *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy*, 2017, p. To appear.

[38] A. Di Sorbo, S. Panichella, C. Alexandru, J. Shimagaki, C. Visaggio, G. Canfora, and H. Gall, "What would users change in my app? summarizing app reviews for recommending software changes," in *FSE 2016*.

[39] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *ICPC 2013*. IEEE, pp. 23–32.

[40] L. F. Cortes-Coy, M. L. Vásquez, J. Aponte, and D. Poshyvanyk, "On automatically generating commit messages via summarization of source code changes," in *SCAM 2014*. IEEE, pp. 275–284.

[41] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *FSE 2014*. ACM, pp. 484–495. [Online]. Available: http://doi.acm.org/10.1145/2635868.2635870

[42] M. Kamimura and G. Murphy, "Towards generating human-oriented summaries of unit test cases," in *Proc. of the International Conference on Program Comprehension (ICPC)*. IEEE, May 2013, pp. 215–218.

[43] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "Who is going to mentor newcomers in open source projects?" in *FSE 2012*, pp. 44:1–44:11.

[44] G. Canfora, M. Di Penta, S. Giannantonio, R. Oliveto, and S. Panichella, "YODA: Young and newcomer developer assistant," in *ICSE 2013*. IEEE CS Press.

[45] S. Panichella, G. Bavota, M. Di Penta, G. Canfora, and G. Antoniol, "How developers' collaborations identified from different sources tell us about code changes." in *ICSME 2014*. IEEE Computer Society, pp. 251–260.

[46] G. Bavota, S. Panichella, N. Tsantalis, M. Di Penta, R. Oliveto, and G. Canfora, "Recommending refactorings based on team co-maintenance patterns," in *Proceedings of the 29th international conference on Automated Software Engineering (ASE 2014)*, 2014.

[47] S. Panichella, G. Canfora, M. Di Penta, and R. Oliveto, "How the evolution of emerging collaborations relates to code changes: an empirical study." in *Proceedings of the 36th International Conference on Program Comprehension*, Hyderabad, India, 2014, pp. 177–188.

[48] G. Bavota, G. Canfora, M. D. Penta, R. Oliveto, and S. Panichella, "An empirical investigation on documentation usage patterns in maintenance tasks," in *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*, 2013, pp. 210–219.

[49] A. D. Lucia, M. D. Penta, R. Oliveto, A. Panichella, and S. Panichella, "Using IR methods for labeling source code artifacts: Is it worthwhile?" in *IEEE 20th International Conference on Program Comprehension, ICPC 2012, Passau, Germany, June 11-13, 2012*, 2012, pp. 193–202. [Online]. Available: https://doi.org/10.1109/ICPC.2012.6240488

[50] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: an empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383–1420, 2014.

[51] S. Panichella, J. Aponte, M. Di Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, June 2012, pp. 63–72.

[52] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, "Codes: Mining source code descriptions from developers discussions," in *ICPC 2014*. ACM, 2014, pp. 106–109. [Online]. Available: http://doi.acm.org/10.1145/2597008.2597799

[53] Y. Zhou, R. Gu, T. Chen, Z. Huang, S. Panichella, and H. C. Gall, "Analyzing apis documentation and code to detect directive defects," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, 2017, pp. 27–37. [Online]. Available: https://doi.org/10.1109/ICSE.2017.11

[54] E. Hill, L. Pollock, and K. Vijay-Shanker, "Automatically capturing source code context of nl-queries for software maintenance and reuse," in *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2009, pp. 232–242.

[55] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving IR-based traceability recovery via noun-based indexing of software artifacts," *Journal of Software: Evolution and Process*, vol. 25, no. 7, pp. 743–762, 2013. [Online]. Available: http://dx.doi.org/10.1002/smr.1564

[56] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, "Codes: Mining source code descriptions from developers discussions," in *ICPC 2014*. ACM, pp. 106–109.

[57] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *ICPC 2012*, 2012, pp. 63–72.

[58] A. D. Sorbo, S. Panichella, C. A. Visaggio, M. D. Penta, G. Canfora, and H. C. Gall, "Development emails content analyzer: Intention mining in developer discussions," in *ASE 2015, pages=12–23, year=2015, organization=IEEE*.

[59] A. Di Sorbo, S. Panichella, C. A. Visaggio, M. Di Penta, G. Canfora, and H. C. Gall, "DECA: development emails content analyzer," in *ICSE 2016, Austin, TX, USA, May 14-22, 2016 - Companion Volume*, 2016, pp. 641–644.

[60] G. Grano, A. D. Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: a dataset for software evolution and quality improvement," in *WAMA@ESEC/SIGSOFT FSE 2017*, pp. 8–11. [Online]. Available: http://doi.acm.org/10.1145/3121264.3121266

[61] S. Panichella, A. Di Sorbo, E. Guzman, C. Visaggio, G. Canfora, and H. Gall, "How can I improve my app? classifying user reviews for software maintenance and evolution," in *ICSME 2015. IEEE International Conference on Software Maintenance and Evolution*, 2015.

[62] A. Ciurumelea, A. Schaufelbuhl, S. Panichella, and H. C. Gall, "Analyzing reviews and code of mobile apps for better release planning," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, 2017, pp. 91–102. [Online]. Available: http://dx.doi.org/10.1109/SANER.2017.7884612

[63] G. Grano, T. Titov, S. Panichella, and H. Gall, "How high will it be? using machine learning models to predict branch coverage in automated testing," in *MaLTeSQuE (Workshop on Machine Learning Techniques for Software Quality Evaluation) 2018 workshop (collocated with SANER 2018), Campobasso, Italy*, 2018, p. To appear.