Crowdsourced Software Development and Maintenance

Bin Lin
REVEAL @ Software Institute – Università della Svizzera italiana (USI)
bin.lin@usi.ch

ABSTRACT

As modern software systems are becoming increasingly complex, developers often need to rely on online sources to address problems encountered during software development and maintenance. These resources provide developers with access to peers' expertise, covering knowledge of different software lifecycle phases, including design, implementation, and maintenance. However, exploiting such knowledge and converting it into actionable items is far from trivial, due to the vastness of the information available online as well as to its unstructured nature. In this research, we aim at (partially) crowdsourcing the software design, implementation and maintenance process by exploiting the knowledge embedded in various sources available on the Web (*e.g.,* Stack Overflow discussions, presentations on SlideShare, open source code, etc.). For example, we want to support software design decisions (*e.g.,* whether to use a specific library for the implementation of a feature) by performing opinion mining on the vast amount of information available on the Web, and we want to recommend refactoring operations by learning from the code written in open source systems. The final goal is to improve developers' productivity and code quality.

ACM Reference Format:
Bin Lin. 2018. Crowdsourced Software Development and Maintenance. In *ICSE '18 Companion: 40th International Conference on Software Engineering , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3183440.3183447

1 RESEARCH PROBLEM AND HYPOTHESIS

Developers often need to search for relevant information from official documentation and/or other online resources, such as tutorials, mailing list, etc. These resources cover knowledge valuable in different phases of the software lifecycle including design, implementation, and maintenance. However, the information provided in these resources can be overwhelming. For example, Stack Overflow[1] currently includes 15 million discussions featuring over 99 million posts (*i.e.,* questions, answers, and comments), among which many report personal opinions about design and implementation choices (*e.g.,* discussions about the best way to implement a given feature in Java). Given the vastness of these online resources, developers often have to spend significant amount of time on extracting and aggregating useful pieces of knowledge from different sources, which results in continuous context switches and reduces their productivity. Additionally, developers might encounter bugs similar to those which others have already fixed or need to write software components implementing features already available in other software projects. Being unable to retrieve and reuse this information undermines the value of online resources and wastes developers' effort.

We aim to propose techniques helping developers to exploit knowledge embedded in various online sources during software development and maintenance activities. More specifically, we will study the possibility of using crowdsourced knowledge to assist developers in (i) taking the best design decisions, (ii) speed up code implementation, and (iii) automatically improve code quality via refactoring operations. We refer to such a perspective as *crowdsourced software development and maintenance.*

The term *crowdsourcing* has been used to refer to *"the act of taking a task traditionally performed by a designated agent (such as an employee or a contractor) and outsourcing it by making an open call to an undefined but large group of people"* [15]. Since the emergence of crowdsourcing, it has been applied in many software engineering contexts. For example, Lim *et al.* [18] developed StakeSource2.0 to identify and prioritize software requirements by automatically creating social networks of stakeholders and asking them to suggest and rate requirements. Ahmed *et al.* [1] built a social computing platform, Jabberwocky, which empowers a human and machine resource management system to distribute programming tasks proposed by developers to the crowd.

Managing the crowdsourcing process remains a big challenge in practice [33]. Indeed (i) it is not easy to identify the people having the right skills for a task to crowdsource, and (ii) the time needed by the crowd to complete the task is difficult to estimate. Therefore, recently some researches [24] have tried to partially automate the crowdsourcing process. For example, Mujumdar *et al.* [24] present an approach to automatically crowdsource solutions to debugging problems. Their basic idea is to leverage test-driven development as a source for bug fixing examples. Thus, they mostly focus on crowdsourced knowledge that can be extracted from code to support code-related activities (*i.e.,* bug-fixing). A similar intuition is also behind the works presenting approaches to support automatic bug-fixing [4, 9, 12, 21, 32]. Still, the value of knowledge embedded in online sources (*e.g.,* developers' opinions about the quality of a code component) is strongly under-exploited.

In our vision of *crowdsourced software development and maintenance*, we want to crowdsource (at least partially) not only code development/maintenance activities such as bug-fixing, but also design decisions. Our approaches should be able to collect and summarize the vast knowledge embedded in online resources, and then convert it into actionable items. More specifically, in this research, we plan to investigate the following research tracks:

[1] www.stackoverflow.com

ICSE '18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05…$15.00
https://doi.org/10.1145/3183440.3183447

Supporting code development and maintenance activities by learning from open source projects. We want to analyze the code of the millions of open source systems available on online forges to (partially) automatize code development and maintenance (*e.g.,* by supporting automatic code completion or recommending refactoring operations).

Supporting software design choices by exploiting crowdsourced knowledge. The goal is to define approaches to mine from online sources opinions expressed by a large group of developers to support design decisions (*e.g.,* implementing a feature from scratch as opposed to reuse an existing library). The opinions must be summarized and converted into pieces of knowledge that can be used to support decision-making processes.

We expect our work to not only have an impact in the software engineering research community by promoting the usage of crowdsourced knowledge, but to also enhance developers' productivity in their everyday activities.

2 CROWDSOURCING CODE-RELATED ACTIVITIES

Source code, like human language, is repetitive rather than unique [11]. Moreover, source code is predictable. This characteristic has been used to recommend code completion candidates by learning from other parts of source code [14]. Therefore, mining source code might be an effective way to enrich the code completion feature. We have conducted a study to understand code redundancy patterns (*i.e.,* where source code tends to be unique as opposed to repetitive) by analyzing a large-scale dataset of active Java projects mined from GitHub [19]. Our results unveil that although code redundancy is common, it is not uniform and mainly resides in specific code constructs (*e.g.,* in `import` statements). We further investigated the implications of the locality of redundancy by analyzing the performance of n-gram language models when used to support code completion [14]. Our study found that while code redundancy can be used for code completion, its locality highly impacts the performance of the language model-based code completion. This finding can serve as a theoretical base for the development of a smarter code completion approach based on information crowdsourced from other systems.

Code completion is only one of the many code-related activities that can be automated by exploiting crowdsourced knowledge. For example, if we are able to learn from developers' good practices (*e.g.,* how to refactor code), we can exploit such a knowledge to automatically improve the quality of our source code that is recognized as one of the major factors determining the success of software projects [27]. In such a context, we investigated the possibility of improving the code quality by learning from other code [20]. More specifically, we focused on recommending rename refactoring operations for identifiers by exploiting the lexical information extracted from source code. Rename refactoring is important because identifiers account for 30% of the tokens and 70% of the characters in the source code [8]. Naming identifiers in a careful, meaningful, and consistent manner likely eases program comprehension and supports developers in building consistent and coherent conceptual models [26].

In the study, we used a n-gram language model to learn good coding practices from a code base and automatically recommend rename refactoring operations aimed at ensuring a consistent use of identifiers in a given software system (*e.g.,* two variables representing the same object in two different classes should be named with the same identifier). We compared our approach with two state-of-the-art techniques, one exploiting static analysis [30], and one using natural language processing [2]. We asked the original developers of five software systems to assess the meaningfulness of the recommendations generated by the three techniques, for a total of 922 rename refactorings manually evaluated. This is the largest empirical study conducted to assess and compare rename refactoring tools. Our approach obtained the best performance among the three tools [20]. The achieved results also shed light on the possibility of crowdsourcing refactoring operations.

3 CROWDSOURCED DESIGN DECISIONS

Developers often consult online sources while taking design decisions (*e.g.,* whether to implement a given feature from scratch as compared to reuse an available implementation). Given the complexity of these choices and their important implications on the success of a software project, it is crucial for developers to take the best decision, possibly in a timely manner. We believe that crowdsourcing opinions from the Web can help them in better dealing with several of these design choices. To start exploring this research area, we worked on the definition of a novel approach to recommend software libraries to developers. Our initial idea was to leverage crowdsourced knowledge on software libraries by mining opinions posted by developers while discussing on Q&A websites such as Stack Overflow. A similar idea has been recently proposed by Uddin and Khomh [31]. Their approach exploits existing sentiment analysis tools to extract the sentiment expressed in the mined opinions (*e.g.,* to classify an opinion about the performance of a library as positive). However, recent studies have found that existing sentiment analysis tools achieve poor accuracy when predicting sentiment on software related textual artifacts [17]. Besides, the system can only provide summarized opinions when developers know exactly which libraries they are going to use, that is, no explicit recommendation is given for a given feature that developers' want to implement.

Our plan is to develop a recommender system as depicted in Fig. 1. The dashed arrows represent dependencies (*e.g.,* ① and ③), while the full arrows indicate flows of information pushed from one component to another. Arrows depicted in red (*i.e.,* those numbered from ① to ⑦) indicate operations performed only once with the goal of storing crowdsourced opinions about software libraries in a database; the black ones represent instead actions triggered by a request for recommendations about the software library to use made by the developer using the front-end.

The system we envision mainly consists of three components: *libraries miner*, *fine-grained linker*, and *opinion miner*.

- *Libraries miner.* The *libraries miner* mines all Java libraries available in maven central[2] (① in Fig. 1) and stores the information in our database ②.

[2]http://central.maven.org/maven2/maven/

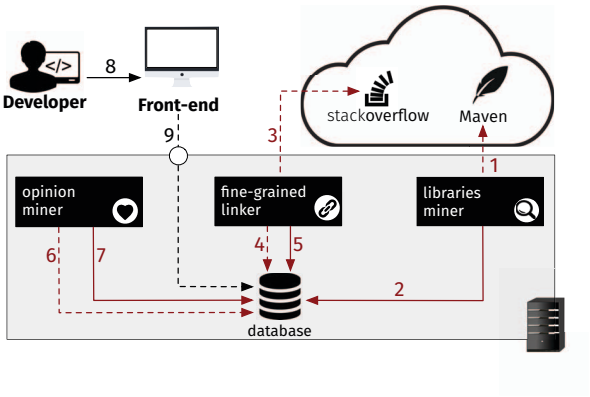

Figure 1: Our vision of the library recommender system.

- *Fine-grained linker.* The *fine-grained linker* mines Stack Overflow discussions to establish fine-grained links between the libraries stored in the database (4) and relevant sentences in Stack Overflow discussions (3). The linking information will also be stored in the database (5).
- *Opinion miner.* The *opinion miner* retrieves the linked sentences (6) for further processing, namely identifying opinions, and then store them into the database (7).

When a developer is interested in receiving recommendations on software libraries, she submits a textual query describing the feature she needs to implement and expresses, on a scale from one to five, the importance of different non-functional requirements (8) (*e.g.*, performance, usability). This information is sent to a Web service (9), to (i) identify libraries possibly relevant for the feature described in the query by using state-of-the-art Information Retrieval (IR) approaches, and (ii) identify the most suitable library considering the non-functional requirements desired by the developer.

One of the fundamental components for the proper working of this recommendation system is the *opinion miner*, that classifies opinions based on their sentiment (positive, neutral, or negative) and on the non-functional requirement (if any) they refer to. Therefore, we started searching for appropriate sentiment analysis tools to adopt.

Sentiment analysis was initially design to classify product reviews [7], and later it has also been used to analyze texts from other domains such as movie reviews [23]. In recent years, sentiment analysis techniques have also been used to analyze online resources in the software engineering domain, such as forum posts [10, 13] and app reviews [5, 6, 22]. In the software engineering field, researchers often use state-of-the-art sentiment tools including SentiStrength [29], NLTK [16], Stanford CoreNLP [28], and EmoTxt [3]. However, as previously said, studies have found that using existing sentiment analysis tools to analyze software engineering related texts can lead to unsatisfactory results. Jongeling *et al.* [17] evaluated the performance of these sentiment analysis tools on a human labeled golden set from a developer emotions study by Murgia *et al.* [25]. The result showed that these tools achieve very poor performance when mining opinions from software engineering datasets, with an accuracy level lower than 50% (meaning that in most of cases, they fail to classify positive/negative sentiments expressed in a set of given sentences). Their further experiment also confirmed that disagreement between these tools can result in contradictory results when using them to conduct software engineering studies.

Therefore, we cannot directly adopt the existing tools to extract opinions from online discussions, instead, we need to train our own model specifically for the software engineering domain. After thorough comparison of different sentiment analysis tools, we decided to adopt one of the most promising opinion mining tool Stanford CoreNLP [28]. We invested a substantial effort in creating a customized training set for it, namely 40k manually labeled sentences/words extracted from Stack Overflow. Despite such a time-consuming training process, the results were negative, with very poor accuracy achieved by the approach in assessing the sentiment of sentences. We also compared our trained tool with all major techniques used in the software engineering community by examining their performance on three different software engineering datasets: Stack Overflow discussions, mobile app reviews, and JIRA issue comments. What we achieved is a bold and negative result: None of the experimented tools, not even the one we explicitly retrained on Stack Overflow, was able to provide a reliable assessment of the opinions mined from software engineering datasets, with an accuracy level close to 50%. The results of this study are under review in a conference adopting double-blind review and, thus, cannot be referenced.

Thus, our plan for the future months is to study how to successfully mine opinions from software engineering datasets. This will be the stepping stone to then build decision-support systems for software developers exploiting crowdsourced knowledge.

4 CONCLUSION AND FUTURE WORK

Our previous work mainly studied the feasibility of our proposed research tracks. More specifically, we investigated the possibility of reusing information from existing code to **support code-related activities** and the possibility of identifying opinions expressed in software-related texts, with the goal of **supporting software design decisions**. For these two research tracks, we propose the following future work:

Supporting code-related activities.

- *Crowdsourced code completion.* Current code completion tools are usually built on datasets containing a limited number of code repositories and always provide the same recommendations in spite of different code contexts. We plan to leverage the code available in open source systems to provide more accurate code completion recommendations in IDEs, also considering the specific code context in which code completion is required.
- *Crowdsourced code quality improvement.* Since we have already proved that refactoring operations can be learned from other code, we will further apply learned rename refactoring operations to automatically generated source code, with the purpose of verifying its usefulness in other development contexts. In the future, we also plan to develop approaches able to learn from other refactoring actions such as reorganizing the code structure.

Supporting software design decisions.

- *Accomplish the software libraries recommender based on mined opinions.* Instead of approaches based on manually labeled sentiment datasets, we will consider alternative options to mine opinions from software engineering datasets. We will evaluate our recommendation system mainly from two perspectives: (i) its ability to correctly identify libraries and corresponding opinions starting from the textual description of the feature to implement provided as input by developers, and (ii) the usefulness for developers during a software design task. For this second point, we plan to conduct a controlled experiment asking developers to pick the best library for a given implementation task given specific non-functional requirements. Different groups of developers will be asked to perform this task for several scenarios with and without the help of our tool. First, we will measure the time needed to make the choice when using/not using the recommendation of our tool. Then, we will discuss with them cases of disagreement (*i.e.,* different library selected manually and with the help of our tool) to collect qualitative feedback.
- *Investigate the use of mined opinions to support other types of design decisions.* If our first task succeeds, we will then investigate the possibility of supporting other design decisions such as those related to data structures to use for representing specific types of data, design patterns to adopt.

I just started my second PhD year, and I plan to spend roughly 15 months on each research track before starting writing my PhD thesis. I believe that the research plan presented in this paper poses a few interesting research problems and hope that it can lead to valuable solutions useful for both academia (*e.g.,* new opinion mining techniques tailored for software engineering datasets) and industry (*i.e.,* new recommendation systems supporting their daily activities).

REFERENCES

[1] Salman Ahmad, Alexis Battle, Zahan Malkani, and Sepander Kamvar. 2011. The Jabberwocky programming environment for structured social computing. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 53–64.

[2] Miltiadis Allamanis, Earl T. Barr, Christian Bird, and Charles Sutton. 2014. Learning Natural Coding Conventions. In *Proceedings of FSE 2014 (22nd International Symposium on Foundations of Software Engineering)*. 281–293.

[3] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2017. EmoTxt: A Toolkit for Emotion Recognition from Text. In *Proceedings of the 7th Affective Computing and Intelligent Interaction*.

[4] Eduardo C Campos, Martin Monperrus, and Marcelo A Maia. 2016. Searching stack overflow for API-usage-related bug fixes using snippet-based queries. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 232–242.

[5] Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. 2014. AR-miner: Mining Informative Reviews for Developers from Mobile App Marketplace. In *Proceedings of ICSE 2014 (36th International Conference on Software Engineering)*. 767–778.

[6] Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald C Gall. 2017. Analyzing Reviews and Code of Mobile Apps for better Release Planning. In *Proceedings of SANER 2017 (24th IEEE International Conference on Software Analysis, Evolution and Reengineering)*. 91–102.

[7] Kushal Dave, Steve Lawrence, and David M Pennock. 2003. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of WWW 2003 (12th Annual International Conference on World Wide Web)*. ACM, 519–528.

[8] Florian Deissenboeck and Markus Pizka. 2006. Concise and consistent naming. *Software Quality Journal* 14, 3 (2006), 261–282.

[9] Thomas Durieux and Martin Monperrus. 2016. Dynamoth: dynamic code synthesis for automatic program repair. In *Proceedings of the 11th International Workshop on Automation of Software Test*. ACM, 85–91.

[10] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason Hong, and Norman Sadeh. 2013. Why people hate your app: Making sense of user feedback in a mobile app store. In *19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1276–1284.

[11] Mark Gabel and Zhendong Su. 2010. A study of the uniqueness of source code. In *Proceedings of FSE 2010 (18th ACM SIGSOFT International Symposium on Foundations of Software Engineering)*. ACM, 147–156.

[12] Qing Gao, Hansheng Zhang, Jie Wang, Yingfei Xiong, Lu Zhang, and Hong Mei. 2015. Fixing recurring crash bugs via analyzing q&a sites (T). In *Proceedings of ASE 2015 (30th International Conference on Automated Software Engineering)*. IEEE, 307–318.

[13] Michael Goul, Olivera Marjanovic, Susan Baxley, and Karen Vizecky. 2012. Managing the Enterprise Business Intelligence App Store: Sentiment Analysis Supported Requirements Engineering. In *Proceedings of HICSS 2012 (45th Hawaii International Conference on System Sciences)*. 4168–4177.

[14] Abram Hindle, Earl T Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. 2012. On the naturalness of software. In *Proceedings of ICSE 2012 (34th International Conference on Software Engineering)*. IEEE, 837–847.

[15] Jeff Howe. 2008. *Crowdsourcing: Why the Power of the Crowd Is Driving the Future of Business* (1 ed.). Crown Publishing Group, New York, NY, USA.

[16] Clayton J Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*.

[17] Robbert Jongeling, Proshanta Sarkar, Subhajit Datta, and Alexander Serebrenik. 2017. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering* (2017), 1–42.

[18] Soo Ling Lim, Daniela Damian, and Anthony Finkelstein. 2011. StakeSource2. 0: using social networks of stakeholders to identify and prioritise requirements. In *Proceedings of ICSE 2011 (33rd International Conference on Software Engineering)*. IEEE, 1022–1024.

[19] Bin Lin, Luca Ponzanelli, Andrea Mocci, Gabriele Bavota, and Michele Lanza. 2017. On the uniqueness of code redundancies. In *Proceedings of ICPC 2017 (25th International Conference on Program Comprehension)*. IEEE Press, 121–131.

[20] Bin Lin, Simone Scalabrino, Andrea Mocci, Rocco Oliveto, Gabriele Bavota, and Michele Lanza. 2017. Investigating the Use of Code Analysis and NLP to Promote a Consistent Usage of Identifiers. In *Proceedings of SCAM 2017 (17th International Working Conference on Source Code Analysis and Manipulation)*. IEEE, 81–90.

[21] Fan Long and Martin Rinard. 2016. Automatic patch generation by learning correct code. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 298–312.

[22] Walid Maalej, Zijad Kurtanović, Hadeer Nabil, and Christoph Stanik. 2016. On the automatic classification of app reviews. *Requirements Engineering* 21, 3 (2016), 311–331.

[23] Gilad Mishne, Natalie S Glance, et al. 2006. Predicting Movie Sales from Blogger Sentiment.. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*. 155–158.

[24] Dhawal Mujumdar, Manuel Kallenbach, Brandon Liu, and Björn Hartmann. 2011. Crowdsourcing suggestions to programming problems for dynamic web development languages. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*. ACM, 1525–1530.

[25] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th working conference on mining software repositories*. ACM, 262–271.

[26] Simone Scalabrino, Mario Linares Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2016. Improving code readability models with textual features. In *ICPC 2016 (24th International Conference on Program Comprehension)*. 1–10.

[27] Kathy Schwalbe. 2015. *Information technology project management*. Cengage Learning.

[28] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of EMNLP 2013 (2013 Conference on Empirical Methods in Natural Language Processing)*.

[29] Mike Thelwall, Kevan Buckley, Georgios Paltoglou, Di Cai, and Arvid Kappas. 2010. Sentiment strength detection in short informal text. *Journal of the Association for Information Science and Technology* 61, 12 (2010), 2544–2558.

[30] Andreas Thies and Christian Roth. 2010. Recommending rename refactorings. In *Proceedings of RSSE 2010 (2nd International Workshop on Recommendation Systems for Software Engineering)*. ACM, 1–5.

[31] Gias Uddin and Foutse Khomh. 2017. Automatic Summarization of API Reviews. In *Proceedings of ASE 2017 (32nd IEEE/ACM International Conference on Automated Software Engineering)*. ACM, 83–94.

[32] Christopher Watson, Frederick WB Li, and Jamie L Godwin. 2012. BlueFix: using crowd-sourced feedback to support programming students in error diagnosis and repair. In *International Conference on Web-Based Learning*. Springer, 228–239.

[33] Shkodran Zogaj, Ulrich Bretschneider, and Jan Marco Leimeister. 2014. Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary. *Journal of Business Economics* 84, 3 (2014), 375–405.