Bridging the Divide Between API Users and API Developers by Mining Public Code Repositories

Maxime Lamothe
Department of Computer Science and Software Engineering
Concordia University
Montreal, Canada
max_lam@encs.concordia.ca

ABSTRACT

Software application programming interfaces (APIs) are a ubiquitous part of Software Engineering. The evolution of these APIs requires constant effort from their developers and users alike. API developers must constantly balance keeping their products modern whilst keeping them as stable as possible. Meanwhile, API users must continually be on the lookout to adapt to changes that could break their applications. As APIs become more numerous, users are challenged by a myriad of choices and information on which API to use. Current research attempts to provide automatic documentation, code examples, and code completion to make API evolution more scalable for users. Our work will attempt to establish practical and scalable API evolution guidelines and tools based on public code repositories, to aid both API users and API developers.

This thesis focuses on investigating the use of public code repositories provided by the open-source community to improve software API engineering practices. More specifically, I seek to improve software engineering practices linked to API evolution, both from the perspective of API users and API developers. To achieve this goal, I will apply quantitative and qualitative research methods to understand the problems at hand. I will then mine public code repositories to develop novel solutions to these problems.

ACM Reference Format:
Maxime Lamothe. 2020. Bridging the Divide Between API Users and API Developers by Mining Public Code Repositories. In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3377812.3382124

1 INTRODUCTION

The current trends of software as a service and open public APIs present increasing opportunities for developers to rely on externally maintained software. However, as a consequence, software developers become dependent on frameworks and public application program interfaces (APIs) [1].

Since the development of an API is typically independent from its usage, API users are at the mercy of the evolution of the interface. The development benefits provided by APIs come at a price. By relying on APIs, users inevitably couple some of their functionality to APIs over which they have little control [2]. This coupling can be a challenge to users as they are forced to deal with ever-evolving APIs [3]. Difficulties with changing dependencies have led to terminology like "dependency hell" [2]. Furthermore, knowledge gaps exist between API developers and API users [4]. Often, API developers communicate about their APIs through one sided documentation channels such as wikis, manuals, tutorials, or API code examples [4]. On the other hand, API users have limited access to API developers and few channels to communicate their feedback. Some APIs even require knowledge of internal politics to reliably get patches accepted [2]. Without a direct feedback channel, situations arise where API developers must rely on repeated user complaints to become aware of existing problems [4]. All too often, when users have issues with an API, for example needing a new feature or experiencing a run-time problem, users may choose to intentionally modify or work around the API [2]. These workarounds allow API users to obtain their desired functionality quickly and without going through potentially arduous communication with API developers. However, these workarounds are not without consequence. On the one hand, since these workarounds are created by API users as temporary solutions, many of these workarounds may become technical debt, endangering code quality and increasing future maintenance cost [5]. On the other hand, these API workarounds are potentially missed opportunities for API developers to improve their APIs (e.g., fixing bugs in the API).

Prior research has mainly concentrated on recommending or producing specialized tools to provide aid to API users. These tools attempt to use existing documentation or historical code-change information to augment API developers' efforts while providing necessary information to API users. Concentrating on API users is particularly prevalent in API migration tools [6]. However, this methodology is often unidirectional, used to extract useful information from API development artifacts to help the API users with tasks such as API migration or API recommendations [1]. API migration and recommendation are not the only API research focuses that concentrate on API users. API misuse research also fixates on API users, concentrating on identifying, detecting, and correcting API usage patterns that can be considered as deviant uses of APIs [7].

In this thesis I propose to bridge the divide between API users and API developers by leveraging public code repositories. Therefore, while I seek to further existing research to provide aid to API users in adapting to API changes, I also seek to keep the API developers

ICSE '20 Companion, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05…$15.00
https://doi.org/10.1145/3377812.3382124

in the development loop. I believe that API users and developers alike can benefit from the knowledge ingrained in public code repositories. To overcome the knowledge gap that exist between them, I propose tools, techniques, and user studies to help API users and API developers. I seek to help API users adapt to changes to APIs (e.g. API migrations). Meanwhile I also seek to help API developers keep up with the ever increasing demands from their users (e.g. API workarounds).

The rest of this paper is organized as follows. Section 2 highlights my research hypothesis. Section 3 aims to help API users through an empirical study of the evolution of the Android API, and a technique to migrate APIs through examples. To help API developers, in Section 4 I propose a study of API workaround patterns and a technique to integrate API modifications into existing APIs. Section 5 surveys related works. Section 6 concludes the paper.

2 RESEARCH HYPOTHESIS

With the advent of open-source software, coupled with the rising popularity of software as a service and APIs, I hypothesize that the divides caused by knowledge gaps between API users and API developers in API development, usage, and maintenance can be bridged by leveraging public code repositories.

The goal of this thesis is to facilitate the evolution, usage, and development of APIs. I believe that by leveraging open-source software through data-mining practices I can provide solutions to problems faced by both API users and API developers. I will leverage open-source repositories to extract historical development information from source code and documentation. Furthermore, I will leverage open discussion forums such as Stack Overflow to uncover real problems faced by developers. Finally, I can further leverage the open-source nature of some APIs to directly propose solutions to problems we find, for example through pull requests, and thereby determine their acceptance by the developer community.

Throughout this thesis I will attempt to bridge the knowledge gap between API developers and API users. For API users I leverage historical data to propose a technique to provide code examples to ease the burden of API migrations. For API developers, I seek to determine patterns and causes for API workarounds and API modifications employed by users. I then seek to develop techniques to reduce API workarounds and API modifications to allow API developers to provide well rounded APIs to their users.

3 HELPING API USERS

In this section, I present research to aid API users. I present an exploratory study into API evolution to understand the problems that plague API users. I also propose an API migration technique to help remedy some of the problems I identify for API users.

3.1 An exploratory study of API evolution

Problem: APIs evolve quickly and require API users to migrate to new versions to benefit from the most cutting edge features [8]. However, API documentation is often reported as incomplete or outdated [9], which makes API migration a challenge for API users. My co-authors and I seek to determine the problems, pitfalls, and most effective solutions to API users that seek API migrations.

How mining public code repositories can help: Online documentation and historical code-changes are prime examples of communication channels between API users and API developers [8].
Our proposed solution: We first leverage the Android documentation, since prior work has shown the importance of documentation in API evolution [10]. As a second step, we leverage historical code change information (e.g. commits) to contrast the results of the API migration pathways obtained from documentation in the previous step. Finally, we use free and open source apps to test the effectiveness of our identified migration pathways; we leverage the API migration suggestions that we automatically recovered from both documentation (including official documentation, commit messages, and code comments) and historical code change information.
Results: We find that the information needed to identify replacement API methods for migrations often resides explicitly in online documentation and repository commits as natural language text. Our findings indicate that although not all migrated methods can be found in the official Android documentation, information needed to assist in API migration can be also found in other forms of documentation, such as code commit messages and code comments.

Our experience agrees with prior research [11] and shows that it is feasible to extract and provide suggestions when migrating API methods to new versions. However, more importantly, we found that implementing API migration code changes is much more challenging than identifying migration pathways. Challenges such as migrating multiple related-APIs, and type changes, present changes which would often require extensive knowledge and effort.

Furthermore, our results and experiences imply that even though documentation is often reported as incomplete or outdated [9], API users should still consider the official documentation of the Android API as their primary source of information. These results show that current practices can allow API users to obtain adequate migration information from API developers to understand *what* to migrate. However, there are still problems in bridging an understanding on *how* to migrate APIs. *Completed and accepted at MSR 2018 [8]*

3.2 API migration through examples

Problem: In today's fast-paced development, APIs are evolving frequently. The Android API is one such example of a rapidly-evolving and widely used API [12]. Such evolution may entail arbitrary release schedules and API deprecation durations, and may involve removing functionality with little to no prior warning [13]. Therefore, users must regularly study the changes to existing APIs and decide whether they need to migrate their code to adopt the changes. As a consequence, there is fragmentation in the user base and slow adopters miss out on new features and fixes [12].
How mining public code repositories can help: Previous research has proposed approaches to help developers with API migration; however, these approaches must be manually pointed towards pre-migrated examples without the ability to automatically retrieve or identify them [14]. Furthermore, the effectiveness of code examples on migration is affected by the context of the examples, whereby examples with closer contexts will waste less developer time when testing extraneous cases [14]. Therefore, by leveraging public code repositories we can consider multiple examples from different contexts to generate well-fitted migration solutions.

Our proposed solution: We propose an approach, named A3, that mines and leverages source code examples to assist developers with API migration. We focus on Android API migrations due to Android's wide adoption and fast evolution [12]. Our approach automatically learns the API migration patterns from code examples taken from available code repositories, thereby providing a variety of example patterns. Afterwards, our approach matches the learned API migration patterns to the source code of the Android apps to identify API migration candidates. If migration candidates are identified, we apply the learned migration patterns to the source code of Android apps, and provide the resulting migration to developers as a potential migration solution.
Preliminary results: To evaluate our approach, first A3 learns Android API migration patterns from three sources of code examples: 1) official Android code examples, 2) migration patterns from the development history of open source Android projects and 3) API migration examples that are manually produced by users. Our approach then applies API migrations to open source Android apps. Our approach can automatically identify 83 migration patterns with 96.1% precision in Android APIs, and obtains a recall of 97% using a seeded repository. Based on 80 migrations candidates in 32 open source apps, our approach can generate 14 faultless migrations, 21 migrations with minor code changes, and 36 migrations with useful guidance to developers. Through a user study with 15 participants and 6 API migration examples, we show that our approach provides, on average, a 29% migration time improvement and is seen as useful by developers. Our approach can be adopted by Android app developers to reduce their API migration efforts to cope with the fast evolution of Android APIs. Our approach also exposes the value of using the knowledge that resides in rich code examples to provide assistance to API users with API related software maintenance.

Expected completion date: early 2020.

4 HELPING API DEVELOPERS

In this section I present avenues of research to help API developers. I present a study into API workarounds to identify new sources of information that can be integrated into APIs by API developers. Furthermore, I also propose a technique to mine public software repositories to extract information that can be used to advise API developers in producing well rounded APIs.

4.1 Studying API workarounds

Problem: As software applications increase in size and complexity, APIs become an integral part of software development. Software is now often produced with help from a slew of APIs to speed up development and reduce project overhead [3]. Dependencies have been shown to be coupled to, and therefore impact, up to 62% of client project source code [15].
How mining public code repositories can help: Open-source repositories and software forums are a rich source of API usage patterns. We can leverage the large number of API uses to determine if workaround pattern instances occur in API usage. Using real API workarounds we can determine their prevalence and the potential impact of integrating a specific workaround as official functionality in the API code base.
Our proposed solution: We conduct an exploratory study of API workarounds requested and implemented by API users. We manually examined 400 posts from Stack Overflow, where we found that API users request API workarounds for a variety of reasons, such as dependency issues, missing functionality, and runtime problems. These reasons illustrate inherent value for API developers since gaining access to these workarounds could improve their APIs. Furthermore, we identified answers accepted by API users who request API workarounds. By studying these answers, we found that carrying out such API workarounds may not be a trivial task. In particular, a majority of API workaround solutions require special implementations to bypass the API.
Preliminary results: To follow up on our exploratory study, we study workaround implementations that are suggested in the Stack Overflow posts, and we observe three generalized API workaround patterns. The knowledge contained in the implementation of these patterns in API user projects can help API developers improve their API by adding desirable unsupported features, fixing unexpected behavior, and improving backwards compatibility.

Since our API workaround patterns were uncovered using forum questions and answers, we seek to confirm their existence in real-life API user code and confirm their usefulness with API developers. Therefore, using five open-source APIs, we detected three patterns of API workarounds in open-source GitHub projects. Finally, we submitted and observed 12 feature requests to developers based on the API workarounds to improve the APIs. Among these requests, five are already closed, and six more have been confirmed by API developers as bugs or missing features. Our study and findings highlight the value of studying API usage as a means to bridge the gap between API developer and API users in order to assist in the development and maintenance of APIs. *Completed and accepted at ICSE 2020 [16]*

4.2 Advising API developers

Problem: If API users produce modifications to APIs for workarounds, then there might exist other more general instances of API modification. These API modifications may contain valuable information that can be used to guide the development and maintenance of APIs. Therefore, we want to determine whether we can automatically leverage user modifications of APIs to determine useful development prospects for API developers. If we can automatically extract useful API development prospects, then we can save time for API developers, and allow desirable features to be integrated into APIs for API users, alleviating their maintenance efforts.
How mining public code repositories can help: Similarly to API workarounds, open-source repositories can provide a rich source of data for API usage patterns. We will therefore seek to leverage a large number of open-source repositories to identify common extension points for APIs and determine if any API extension patterns occur in the API ecosystem. Based on these patterns we will identify prospects for API improvement for API developers.
Our proposed solution: To answer this research question, we will create a tool to parse Java repositories to detect user modifications to existing APIs. We will gather a representative sample of API user applications from open-source hosting services like GitHub, and gather data for several Java based open-source APIs. We will

attempt to automatically determine potential extension points for APIs that do not currently exist as part of the API.

Our proposed evaluation: We will provide our mined API modifications to API developers as pull requests or feature requests to determine whether they are useful to API developers or not.

Expected completion date: late 2020 - early 2021.

5 STATE OF THE ART AND PRACTICE

The following research is closely related to the work in this thesis.

API migration and suggestion tools Previous research has produced numerous API migration mapping methods [17], API studies and API suggestion approaches [18] that rely on historical source code repositories [9] or documentation [19]. Some suggestion approaches like ApiRec produced by Nguyen et al. [1] are based on the intuition that developers make low-level changes while having a higher-level intent in mind. However, these tools rely on historical source-code information and existing documentation, they make the assumption that the information they seek can be retrieved from historical data, or the APIs source code. These tools often take API users out of the API development cycle. Furthermore, the migration methods usually present a single best migration pathway [9] and may not present migration pathways that require multiple APIs.

Code examples and APIs *EXAMPLORE* is a visualization tool to assist users in understanding common uses of APIs [20]. It uses API examples to build common usage patterns. Since tools like *EXAMPLORE* have shown useful in assisting users with API questions [20], I believe that using examples to provide API insights could similarly be used for migration tools.

API usage and misuse API usage patterns have been used in prior work to understand how APIs evolve [12], to detect API compatibility issues [21], for API migration [8], code recommendation [22] and more. API usage patterns can be extracted from source code [8] and examples [20]. API misuses are characterized by their violation of API constraints [7]. Misusing APIs can lead to an increase in software security vulnerabilities and bugs [7]. This thesis concentrates on constructive workarounds that could be API improvements, and not on API misuses or unintentional alternate uses of APIs. I specifically concentrate on API usage that would not typically be expected from API developers (i.e. workarounds).

6 CONCLUSION

This proposal centers around mining public code repository data to improve API software engineering practices. More specifically, I focus on bridging the divide between API users and API developers. I seek to improve API usability for API users by understanding the problem of API migration and reducing the migration effort required from API users. Through preliminary findings, I have shown that API migration is not a simple problem, but that existing API migrations can be used as examples to help future migrations. I seek to understand why and how API users use API workarounds to help API developers produce well rounded APIs. I have shown that an understanding of API workaround practices can be used to inform API developers to reinforce the feedback loop between API developers and API users. My future work will concentrate on further improving API engineering practices by leveraging user modifications of APIs to advise API development. My contributions include: 1) a novel API migration technique for API users; 2) opening research avenues in API workarounds; 3) providing automated suggestions of improvements for API developers. I expect to finish the work for this thesis in 2021.

REFERENCES

[1] A. T. Nguyen, M. Hilton, M. Codoban, H. A. Nguyen, L. Mast, E. Rademacher, T. N. Nguyen, and D. Dig, "Api code recommendation using statistical learning from fine-grained changes," *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, 2016.

[2] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "How to break an api: Cost negotiation and community values in three software ecosystems," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, (New York, NY, USA), pp. 109–120, ACM, 2016.

[3] D. Dig and R. Johnson, "How do apis evolve? a story of refactoring," *Journal of Software: Evolution and Process*, vol. 18, no. 2, pp. 83–107, 2006.

[4] B. Dagenais and M. P. Robillard, "Recovering traceability links between an api and its learning resources," *2012 34th International Conference on Software Engineering (ICSE)*, 2012.

[5] A. Potdar and E. Shihab, "An exploratory study on self-admitted technical debt," in *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*, pp. 91–100, 2014.

[6] B. E. Cossette and R. J. Walker, "Seeking the ground truth: A retroactive study on the evolution and migration of software libraries," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, (New York, NY, USA), pp. 55:1–55:11, ACM, 2012.

[7] S. Amann, H. A. Nguyen, S. Nadi, T. N. Nguyen, and M. Mezini, "Investigating next steps in static api misuse detection," in *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR 19, pp. 265–275, 2019.

[8] M. Lamothe and W. Shang, "Exploring the use of automated api migrating techniques in practice: An experience report on android," *MSR '18: 15th International Conference on Mining Software Repositories*, 2018.

[9] B. Dagenais and M. P. Robillard, "Recommending adaptive changes for framework evolution," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 4, pp. 1–35, 2011.

[10] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vasquez, and et al., "On-demand developer documentation," *2017 IEEE International Conference on Software Maintenance and Evolution*, 2017.

[11] T. D. Nguyen, A. T. Nguyen, and T. N. Nguyen, "Mapping api elements for code migration with vector representations," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pp. 756–758, 2016.

[12] T. Mcdonnell, B. Ray, and M. Kim, "An empirical study of api stability and adoption in the android ecosystem," *2013 IEEE International Conference on Software Maintenance*, 2013.

[13] A. A. Sawant, R. Robbes, and A. Bacchelli, "On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK," *Empirical Software Engineering*, pp. 1–40, 2017.

[14] M. P. Robillard, W. Maalej, R. J. Walker, and T. Zimmermann, eds., *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, 2014.

[15] G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella, "How the apache community upgrades dependencies: An evolutionary study," *Empirical Softw. Engg.*, vol. 20, pp. 1275–1317, Oct. 2015.

[16] M. Lamothe and W. Shang, "When apis are intentionally bypassed: An exploratory study of api workarounds," in *Proceedings. 42nd International Conference on Software Engineering*, ICSE 2020, 2020.

[17] M. Kim, "LASE: Locating and Applying Systematic Edits by Learning from Examples," *ICSE*, pp. 502–511, 2013.

[18] T. Zhang, D. Yang, C. Lopes, and M. Kim, "Analyzing and supporting adaptation of online code examples," in *41st International Conference on Software Engineering*, ICSE '19, (Piscataway, NJ, USA), pp. 316–327, IEEE Press, 2019.

[19] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Jdiff: A differencing technique and tool for object-oriented programs," *Automated Software Engineering*, vol. 14, no. 1, pp. 3–36, 2006.

[20] E. L. Glassman, T. Zhang, B. Hartmann, and M. Kim, "Visualizing api usage examples at scale," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, (New York, NY, USA), pp. 580:1–580:12, ACM, 2018.

[21] S. Scalabrino, G. Bavota, M. Linares-Vásquez, M. Lanza, and R. Oliveto, "Data-driven solutions to detect api compatibility issues in android: An empirical study," in *Proceedings of the 16th International Conference on Mining Software Repositories*, MSR '19, (Piscataway, NJ, USA), pp. 288–298, IEEE Press, 2019.

[22] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, and M. Di Penta, "Focus: A recommender system for mining api function calls and usage patterns," in *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, (Piscataway, NJ, USA), pp. 1050–1060, IEEE Press, 2019.