Why Do We Break APIs? First Answers from Developers

Laerte Xavier, Andre Hora, Marco Tulio Valente
ASERG Group, Department of Computer Science (DCC)
Federal University of Minas Gerais (UFMG), Brazil
{laertexavier,hora,mtov}@dcc.ufmg.br

***Abstract*—Breaking contracts have a major impact on API clients. Despite this fact, recent studies show that libraries are often backward incompatible and that the rate of breaking changes increase over time. However, the specific reasons that motivate library developers to break contracts with their clients are still unclear. In this paper, we describe a qualitative study with library developers and real instance of API breaking changes. Our goal is to (i) elicit the reasons why developers introduce breaking changes; and (ii) check if they are aware about the risks of such changes. Our survey with the top contributors of popular Java libraries contributes to reveal a list of five reasons why developers break API contracts. Moreover, it also shows that most of developers are aware of these risks and, in some cases, adopt strategies to mitigate them. We conclude by prospecting a future study to strengthen our current findings. With this study, we expect to contribute on delineating tools to better assess the risks and impacts of API breaking changes.**

***Index Terms*—Software Evolution; API Stability; Backwards Compatibility; Qualitative Study.**

I. Introduction

Despite of the risks for client applications, breaking API contracts is a common practice. While maintaining libraries, developers often introduce breaking changes (*e.g.,* removing public classes, fields, or methods) [1]–[4]. In a recent study [5], conducted at a large scale level, we concluded that 28% of all API changes break backward compatibility (on the median, 14.78% of changes per library).

However, there is a limited number of studies investigating the real motivations driving API breaking changes. For instance, Dig and Johnson [6] manually analyzed API change logs and release notes to identify the types of existing breaking changes. However, they focus on providing a catalog of API changes to better understand the requirements of API migration tools. Bogart *et al.* [7] performed a general-purpose case study with 28 developers to study how they plan, manage, and negotiate breaking changes. However, they report developers general views and conceptions on these changes. Therefore, we are still unaware about the specific reasons that motivate breaking changes in open source projects. For example, these are questions lacking clear answers: (a) Why do API developers, who are supposed to be careful about compatibility, break API contracts? (b) When this happens, are they aware of the risks to client applications?

In this paper, we describe a qualitative study with library developers and real instances of API breaking changes. Specifically, our goal is to elicit a list of motivations for API breaking changes from library developers and verify their awareness on the impact on client applications. Thus, we propose the following research questions:

- RQ1. Why do developers break API contracts?
- RQ2. Are developers aware of the impact of breaking changes on client applications?

To answer these questions, we conducted a survey with top Java library developers. Our results show a list of five reasons why developers break API contracts. Moreover, we also show that most developers are aware of the risks, and, in some cases, they also adopt strategies to alleviate the impact on client applications (*e.g.,* deprecation). Based on these results, we suggest a future study to strengthen our current findings and also to support the development of tools to better assess the risks and impacts of API breaking changes. Thus, the contributions of this paper are summarized as follows:

- We provide a qualitative study to elicit the motivations of API breaking changes and to understand developers concerns with the impact on client applications.
- We prospect a study based on *firehouse interviews* [8] to strengthen our current findings.

Structure of the paper: Section II presents the previous study that motivates the current work. In Section III, we describe our qualitative analysis by detailing the survey design, the results, and the threats to validity. Related work is discussed in Section IV. Finally, Section V concludes by prospecting a future study to strengthen our current findings.

II. Previous Study

In our previous study [5], we investigated at a large-scale level API breaking changes and their impact on client applications. Specifically, two major points were addressed: (i) the frequency of API breaking changes, and (ii) the behavior of these changes over time. We present below a brief description of our previous study, as it motivated the survey presented in Section III.

Following the definitions of Dig and Johnson [6], we classified as *breaking changes* the code modifications in all API elements that break backward compatibility (*i.e.,* removal of types, fieds, and methods; visibility tightening; and signature changes). To analyze the frequency of these changes in Java libraries, we implemented an API *diff* tool based on Eclipse

978-1-5090-5501-2/17/$31.00 © 2017 IEEE

JDT. Thus, we collected a set of 317 popular libraries hosted on GitHub and performed our analysis as follows.

In a first analysis, we focused on the frequency of API breaking changes. We analyzed the frequency of changes for types, fields, and methods between the two latest releases of 317 Java libraries. From these libraries, 198 (62.46%) have at least one breaking change. In total, 140,460 breaking changes were identified; methods are the API elements with more changes. Considering the absolute distribution of the number of breaking changes per library, the first quartile is 0, the median is 4, and the third quartile is 75. Outlier values are very different: we detect a library with 11,816 breaking changes for fields, and another one with 1,392 breaking changes for types, which were explained by large structure and design changes. Considering the relative distribution, we detected that the first quartile is 0%, the median is 14.78%, and the third quartile is 43.35%. In short, libraries often break backward compatibility. We observe that API breaking changes are recurrent and occur in a relevant percentage.

In our second experiment, we investigated the behavior of API breaking changes over time to better understand whether library stability tends to get better (or worse). In this case, we analyzed *all* releases of the 317 studied libraries. To accomplish that, we verify 9,329 releases and summarize the frequency of breaking changes per year. The number of analyzed libraries per years is 232, 212, 149, 106, and 83. The median of breaking changes per library is 29.02%, 31.46%, 37.12%, 45.16%, and 49.14%. Thus, the historical analysis of the breaking changes frequency show that they increase by 20% in five years. This result shows that as time passes, libraries do not become more stable, as expected.

In summary, our previous analysis presents that libraries are often backward incompatible and that the rate of breaking changes increases over time. However, the specific reasons that motivate library developers to break contracts with their clients are still unclear.

III. Survey With Developers

A. Survey Design

To investigate the reasons why developers break API contracts, we performed a survey with the major contributors of the most popular Java libraries hosted on GitHub. We summarized this study in the following steps.

1. Selecting Surveyed Developers. First, we selected the top repositories with more than *50 breaking changes* collected in the study described in Section II. Out of 317 studied libraries, 90 (28.39%) filled this selection criteria. Then, we accessed each repository with the purpose of retrieving the email address of their major contributor. From the total, 49 contributors (54.44%) shared their email on GitHub. Therefore, our initial dataset consists of the corresponding *49 libraries*,[1] including well-known and worldwide used projects, such as BITCOINJ/BITCOINJ, JAVASLANG/JAVASLANG, and JUNIT-TEAM/JUNIT4.

2. Contacting Developers. For each selected library, we sent an email to the corresponding major contributor (between November 12th and 25th 2016). Figure 1 presents the sent email, as well as the proposed questions. In each email, we presented the number of collected breaking changes and an external link describing each of them.[2] Moreover, we proposed three questions with the goal of (i) verifying whether developers are aware about the listed breaking changes; (ii) investigating their reasons for implementing breaking changes; and (iii) verifying whether they are aware about the risks of breaking changes to client applications. Specifically, the first question was used as a filtering criterion in Step 3 (*i.e.,* developers who said not being aware of the breaking changes had their emails discarded).

> *Dear [developer name],*
>
> *I figured out that you are a major contributor of [*REP/PROJECT*], from which we found [n] API breaking changes, for instance, in classes A and B (further details here [link]).*
>
> *I kindly ask you to answer the following questions to support our research:*
>
> *1. Are you aware about these API breaking changes?*
> *2. Could you describe why were these API breaking changes introduced?*
> *3. Are you aware about the risks of breaking backward compatibility with your clients?*

Fig. 1. Email sent to the major contributors of the studied libraries

3. Filtering Responses. We received 14 answers, which represents a response rate of 28.6%. From these answers, 6 were considered unclear or invalid (*e.g.,* responses reporting that the developer is no longer engaged with the project). Besides, one developer stated that he was not aware of the breaking changes (first question), and thus his answer was also discarded. As a result, *7 answers* were considered for analysis. Table I describes the libraries whose responses were analyzed, as well as basic information about them (*i.e.,* number of stars and total of breaking changes). The number of stars ranges from 857 (BITCOINJ/BITCOINJ) to 1,568 (MOGOBD/MONGO-JAVA-DRIVER). The number of breaking changes ranges from 53 (OBLAC/JODD) to 3,117 (MOGOBD/MONGO-JAVA-DRIVER).

TABLE I
LIBRARIES WITH VALID ANSWERS

	Library	Stars	Breaking Changes
D1	MONGODB/MONGO-JAVA-DRIVER	1,568	3,117
D2	OBLAC/JODD	1,445	53
D3	ZIELONY/CARBON	1,380	358
D4	SQUARE/ASSERTJ-ANDROID	1,354	2,218
D5	JAVASLANG/JAVASLANG	1,108	663
D6	DAVIDEAS/FLEXIBLEADAPTER	975	157
D7	BITCOINJ/BITCOINJ	857	1,940

[1]Full dataset description at: https://goo.gl/NWk9x6

[2]All lists of breaking changes sent are available at: https://goo.gl/Gx87JL

4. Analyzing Data. After collecting and filtering all emails, we analyzed the responses in order to investigate the proposed research questions. To answer *RQ1*, we followed a thematic analysis, which is a technique whose goal is to identify themes (or codes) within a set of documents [9]. Thus, the first author of the paper manually analyzed each response to the second question in the survey email and cataloged reasons that may explain why developers break API contracts. The second and third authors reviewed the analysis and confirmed the proposed codes. Finally, to answer *RQ2*, we analyzed responses to the third question, and, as a result, we identified insights on how developers deal with the impact of API breaking changes.

B. Survey Results

RQ1. Why do developers break API contracts?

We identified *five* main reasons that suggest the intents of developers and their recurrent explanations for breaking APIs. Table II describes these reasons, a brief description, and the number of occurrences for each of them.

TABLE II
REASONS WHY DEVELOPERS BREAK API CONTRACTS

Theme	Description	Occur.
LIBRARY SIMPLIFICATION	Redesign to make APIs easier for clients	3
REFACTORING	Remodularization to improve internal code	2
BUG FIX	Resolution of issues	2
DEPENDENCY CHANGES	Switch of libraries on which the library depend	1
PROJECT POLICY	Maintenance policy of the project	1

The most frequent reason for breaking API contracts is related to LIBRARY SIMPLIFICATION (3 instances). In this case, the change is mainly motivated by the need of making APIs easier to use (*e.g.,* developer-friendly code); and also by the remotion of redundant (and more complex) elements. For instance, both developers D2 and D3 mentioned this motivation:

"The change leads to better and more developer-friendly code (for example, to more fluent code)." [D2]

"Useless item. If a problem can be solved using another simple method, the library can be simplified by removing the redundant solution." [D3]

Differently from the LIBRARY SIMPLIFICATION theme, whose intents are related to improving the library for external clients, the second most frequent motivation relates to REFACTORING (2 instances). In this case, developers pointed the need of internally improving the code of their APIs (*e.g.,* by moving elements between packages).

Other studies also indicate REFACTORING as a reason for breaking changes. For example, Dig and Johnson [6] found that 80% of the breaking changes are due to refactoring. D6 illustrates this motivation, detailing a refactoring on his library with the purpose of better organizing package signatures:

"The classes/methods/fields are not removed all, they are just refactored to a better package signature (many months ago/last year) when the library was know a little but not famous as now... When possible they are initially deprecated and then removed completely." [D6]

In addition, developer D4 discussed another motivation related to changing the library dependencies: DEPENDENCY CHANGES. According to him, the breaking changes reported were caused by the need of changing one library that they depend on and was not being maintained anymore:

"We switched the assertion library on which the library was based since FEST was no longer being developed and AssertJ was a maintained and updated fork." [D4]

Finally, other developers commented that the breaking changes are motivated by the need of fixing bugs reported by clients (developers D2 and D3), and due to a deliberated project maintenance policy (developer D6). The first one is illustrated in the following comment:

"Bugfix. For example some of the items shouldn't be accessible and were made private." [D3]

RQ2. Are developers aware of the impact of breaking changes on client applications?

To verify whether developers are aware about the risks of breaking APIs, we analyze the answers to the third question proposed in our initial email. Out of the seven responses, in *five* instances developers affirmed being aware of the risks. In the two remaining, we identified unclear or vague answers. Thus, *all* developers who gave valid responses recognized being aware of the impact and costs of breaking changes. In some cases, they also discussed alternatives to mitigate them.

A first and natural alternative to decrease the impact of breaking changes is to use `deprecated` annotations and replacement messages [2], [4]. Both developers D3 and D2 cited this strategy. However, developer D2 discusses the lack of human resources to maintain deprecated methods.

"I always try to mark things I would like to remove as deprecated, give replacements and document changes to make transitions easy." [D3]

"Once one client asked to use @Deprecated on old methods, but we simply dont have enough resources to maintain all deprecated methods." [D2]

In addition, both developers justified their breaking changes by highlighting the small number of clients of their libraries. Our previous study (Section II) confirms this fact. The library maintained by D3 has no client affected by the collected breaking changes, while only one class of D2's library impacts 7 clients (which represents 9% of the total). The following comments illustrate their statements:

"Carbon is not a commercial, production-quality library, so I'm not as concerned about potential problems as Google is

with their libraries. I'm just working on my ideas and I'm giving my solutions to the public." [D3]

"Yes. But we are not Spring yet. [...] Being a small-to-middle library has it's benefits." [D2]

Finally, developer D4 illustrated an interesting strategy to mitigate the risks of breaking changes. With the purpose of rebuilding the library due to DEPENDENCY CHANGE reasons, and reusing most of the initial code, the decision was to create a new library in the Maven Central Repository.[3] Thus, clients interested in migrating had to switch libraries (and update their code to the new API contracts). This is illustrated in the following comment:

"From the consumer perspective it's a totally different library, not just a new version of an existing one that has a new API. In order to upgrade, consumers would have to change their build to point at the new coordinates. If all they were doing was looking for a new version of the old coordinates they would never see it. Additionally, because it's separate coordinates you can even have both versions installed side-by-side and do incremental migration. We decided to keep the same repository despite essentially creating a new library because they solve the same problem, we could re-use 90% of the code, and there never would be releases made of the old version once we switched." [D4]

C. Summary

In this section, we summarize the results and answer the investigated research questions.

RQ1. Why do developers break API contracts? We elicit a list of *five* specific reasons pointed by developers as motivation for API breaking changes: LIBRARY SIMPLIFICATION, REFACTORING, BUG FIX, DEPENDENCY CHANGES, and PROJECT POLICY. Some of them were recurrent between respondents. For instance, LIBRARY SIMPLIFICATION was discussed in three out of seven analyzed answers. This may reveal that developers are more concerned about the usability of their APIs, despite the possible costs caused by breaking changes.

RQ2. Are developers aware of the impact of breaking changes on client applications? Our study shows that *all* developers are aware of the risks attached to breaking contracts with clients. However, in most of the cases, they highlighted the adoption of alternatives to mitigate them. This result suggests that developers have conscious about the costs for client applications but rather than planning changes and deprecating elements, they prefer adopting strategies to alleviate their side-effects.

D. Threats to Validity

The results presented in this paper provide initial insights on the reasons why developers break APIs and whether they are aware of the consequences for client applications. Despite a response rate of 28.6%, only seven answers were considered for analysis, which impacts the generalization of our results. However, the studied libraries are representative especially due to their popularity (*i.e.,* at least 857 stars), and high number of breaking changes (*i.e.,* more than 50 ones). Another threat is related to the manual inspection of the answers to provide the reasons for breaking changes. Although this activity has been done with special attention and support of the second and third authors, its subjective nature may bias the presented results. Finally, against our belief, the trustworthiness of the responses may also be a threat to be reported.

[3]https://maven.apache.org

IV. RELATED WORK

There are several studies in the literature that focused on analyzing API evolution and stability. In a previous study [5], we perform a large scale investigation with 317 Java libraries and more than 260K client applications. As described in Section II, we found that libraries are often backward-incompatible and that the rate of breaking changes increases over time. Raemaekers *et al.* [10] investigate API stability by defining four metrics based on method changes and removals. Moreover, Jezek *et al.* [11] focus on more subtle issues related to source and certain types of behavioral compatibility. They analyze 109 Java programs and 564 program versions, concluding that API instability is a common phenomenon. Finally, McDonnell *et al.* [12] investigate API stability and adoption of Android libraries and find that the evolution rate of such APIs is faster than clients' update velocity.

As a main strategy to reduce costs of breaking changes, some studies focus on the use of `deprecated` annotations. For instance, Robbes *et al.* [2] show that some API deprecation have large impact on the Smalltalk ecosystem and that replacement messages usually have low quality. Hora *et al.* [3] analyze the impact of API replacement and improvement messages on a large-scale Smalltalk ecosystem. The results show that a large amount of clients are affected by API changes but most of them do not react to them. In a recent work, Brito *et al.* [4] investigate the usage of deprecation messages and provide the basis for a recommendation tool.

However, few studies investigate the reasons that motivate API breaking changes. As an initial effort, Dig and Johnson [6], defined a catalog of *breaking changes* and *non-breaking changes*, and manually analyzed libraries change log and release notes to identify the reasons of breaking changes. As a result, they found that 80% of such changes are refactorings. In this study, we apply this classification of changes and we also identified REFACTORING as a motivation for breaking API contracts. However, we discovered other four reasons for breaking changes. Indeed, the most common reason identified in our study (LIBRARY SIMPLIFICATION) is not discussed by Dig and Johnson.

Finally, Thung *et al.* [7] perform a general-purpose case study to understand how developers reason and apply changes in three software ecosystems: Eclipse, R/CRAN, and Node.js/npm. As a result, they report the differences in practice, polices, and tools applied when performing/avoiding a breaking change. They conclude that in Eclipse developers

do not break APIs; in R/CRAN, they reach out affected clients; and in Node.js/npm, they increase the major version number. Although this is also a qualitative study, we observe that the conclusions stated by the authors do not reflect developers explanations about concrete API breaking changes. Instead, they reflect general perceptions and views about breaking changes in the considered software ecosystems. By contrast, we base our analysis on reasons for concrete breaking changes collected in a period close to the survey.

Novelty: To the best of our knowledge, this is the first study investigating the motivations behind API breaking changes based on the actual explanations of developers on specific breaking changes they have recently applied.

V. Conclusions and Future Work

In this paper, we performed a qualitative study about API breaking changes. We applied a survey with the top developers of popular Java libraries in order to (i) elicit the reasons why developers implement breaking changes, and (ii) check whether they are aware about the risks of these changes. We proposed a list of *five* reasons that motivate developers to break API contracts, including: Library Simplification, Refactoring, Bug Fix, Dependency Changes, and Project Policy. Moreover, we showed that developers are usually aware of the impact on clients and, in some cases, adopt strategies to alleviate them.

As a next step, we plan to perform an in-depth study based on *firehouse interviews* [8] with the contributors of popular Java libraries hosted on GitHub. We intend to replicate a methodology used in a previous study about refactoring motivations [13]. During several months, we plan to monitor a large dataset of libraries, fetching commits from each remote repository to a local copy. Next, we will use the *diff* tool implemented in our previous work [5] to iterate through each commit and identify changes that break compatibility. Finally, an email will be sent to the author of the commit asking two main questions: (a) Could you describe why did you perform the listed breaking change? (b) Are you aware of the possible impact of them in case they are released to clients? Our goal is to contact API developers as soon as they introduce a breaking change, while the change is still fresh. In this way, we plan to receive more answers, which is important to increase confidence on the initial list of reasons for breaking changes elicited in this work. In addition, we also plan to study breaking changes in other popular programming languages, such as JavaScript.

Acknowledgment

We thank all 14 developers for sharing their ideas and answering our emails. Also, we thank CNPq and FAPEMIG for supporting this research.

References

[1] W. Wu, Y.-G. Gueheneuc, G. Antoniol, and M. Kim, "Aura: a hybrid approach to identify framework evolution," in *32nd International Conference on Software Engineering (ICSE)*, 2010, pp. 325–334.

[2] R. Robbes, M. Lungu, and D. Röthlisberger, "How do developers react to API deprecation? The case of a Smalltalk ecosystem," in *20th International Symposium on the Foundations of Software Engineering (FSE)*, 2012, pp. 1–11.

[3] A. Hora, R. Robbes, N. Anquetil, A. Etien, S. Ducasse, and M. T. Valente, "How do developers react to API evolution? The Pharo ecosystem case," in *31st IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2015, pp. 251–260.

[4] G. Brito, A. Hora, M. T. Valente, and R. Robbes, "Do developers deprecate APIs with replacement messages? A large-scale analysis on Java systems," in *23rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2016, pp. 360–369.

[5] L. Xavier, A. Brito, A. Hora, and M. T. Valente, "Historical and impact analysis of API breaking changes: A large-scale study," in *24rd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 1–10.

[6] D. Dig and R. Johnson, "How do APIs evolve? A story of refactoring," in *22nd International Conference on Software Maintenance (ICSM)*, 2006, pp. 83–107.

[7] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, "How to break an API: cost negotiation and community values in three software ecosystems," in *24th Symposium on the Foundations of Software Engineering (FSE)*, 2016, pp. 109–120.

[8] E. Murphy-Hill, T. Zimmermann, C. Bird, and N. Nagappan, "The design space of bug fixes and how developers navigate it," *Transactions on Software Engineering*, vol. 41, no. 1, pp. 65–81, 2015.

[9] D. S. Cruzes and T. Dyba, "Recommended steps for thematic synthesis in software engineering," in *5th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2011, pp. 275–284.

[10] S. Raemaekers, A. van Deursen, and J. Visser, "Measuring software library stability through historical version analysis," in *28th International Conference on Software Maintenance (ICSM)*, 2012, pp. 378–387.

[11] K. Jezek, J. Dietrich, and P. Brada, "How Java APIs break–an empirical study," *Information and Software Technology*, vol. 65, no. C, pp. 129–146, 2015.

[12] T. McDonnell, B. Ray, and M. Kim, "An empirical study of API stability and adoption in the Android ecosystem," in *29th International Conference on Software Maintenance (ICSM)*, 2013, pp. 70–79.

[13] D. Silva, N. Tsantalis, and M. T. Valente, "Why we refactor? Confessions of GitHub contributors," in *24th International Symposium on the Foundations of Software Engineering (FSE)*, 2016, pp. 858–870.