API Governance at Scale

API Governance at Scale

Mak Ahmad
UC Davis
Davis, CA, USA
shahmad@ucdavis.edu

JJ Geewax
Meta
Singapore
jgeewax@meta.com

Andrew Macvean
Google
Seattle, USA
amacvean@google.com

David Karger
MIT
Cambridge, MA, USA
karger@mit.edu

Kwan-Liu Ma
UC Davis
Davis, CA, USA
klma@ucdavis.edu

Abstract

API Governance, the process of applying standardized sets of policies and guardrails to the design and development of APIs, has only grown in importance and prominence given the continued growth in APIs being produced. In this paper, we present an Action Research style approach to investigate and understand the utility of a multi-faceted API Governance process being adopted inside Google. We first reflect on past research around API Governance, and then introduce three new components, 1. API Improvement Proposals (AIPs) the documented source of truth for API design rules, 2. API Linter, an automated analysis tool which checks for adherence to / violations of AIPs, and 3. API Readability, a program to educate and certify API design experts. These three components are designed to build upon pre-existing processes to scale and improve API design. Through a mixed-methods research strategy, containing both a survey and a series of interviews, we evaluate the utility of these approaches in supporting API Producers. Our research shows that API Producers have positive sentiment towards API Governance, validating the general direction of the program. Specifically, our study participants highlighted the positive impact of API Governance on the quality of the APIs they produced, via consistency in both the outcome and approach. This paper also discusses future research opportunities to enhance API Governance, specifically with regards to newer API Producers, who reported worse sentiment towards the program than their more experienced peers.

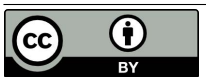
CCS Concepts

• **Software and its engineering → Software development methods**; *Software design engineering*.

Keywords

API governance, API design

ACM Reference Format:
Mak Ahmad, JJ Geewax, Andrew Macvean, David Karger, and Kwan-Liu Ma. 2024. API Governance at Scale. In *46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3639477.3639713



ICSE-SEIP '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0501-4/24/04.
https://doi.org/10.1145/3639477.3639713

1 Introduction

Application Program Interfaces (APIs) are ubiquitous in modern software development. "APIs including libraries, frameworks, toolkits, and software development kits, are used by virtually all Code" [19]. The users/consumers of APIs are developers [2]. APIs expose functionality and data, facilitating developer productivity by preventing costly errors, facilitating code re-use, and more.

Despite their unquestionable importance, APIs are often difficult to use. For example, Eagle and colleagues found that API misuse is widespread, with 88% of applications in the Google Play store having at least one API usage mistake [5]. Unusable APIs not only impact the productivity of the user (e.g., by increasing the time it takes to learn how to use them effectively), but can also contribute to software bugs and security problems [19].

As a result, there is an active field of research around API Usability. Prior research has identified numerous causes of low API usability, including issues with the semantic design of the API, the level of abstraction, the quality of the documentation, the ability to effectively handle errors, and unclear dependencies [15][23][22]. One complicating factor with APIs is that it is difficult to change an API substantially once it is released, as one has the potential to break any software which depends on it. This makes the design process critical.

API designers must make a number of decisions during the design process, for example, which patterns to adopt or naming conventions to utilize [24]. Researchers have explored various techniques to uncover, understand, and measure API usability including, but not limited to, surveys [20], peer reviews [16], lab experiments [25] , and heuristic evaluations [4], all with the goal of aiding the API designer in making informed decisions during the design process. In addition to the aforementioned research methodologies, API designers can also review the similarities and differences of API Style Guides [17] across 32 companies. Despite all of this, the process of designing an API remains complicated and challenging [18] especially when it comes to enforcement of consistency and proven evolvability.

In 2016, Google shared their work on scaling up API Design Reviews as a way to improve both the quality of Google's APIs and the satisfaction of the API designers working on them [16]. The early feedback on their "Apiness" process was positive, but the

paper also foreshadowed a number of areas for future consideration/improvement, in particular given the ever-changing context of building APIs at Google. They state:

"how can we continue scaling the API design review program to not only review more APIs, but also reduce turnaround time, all while striving to maintain the quality of the review process? ... Additionally, with a growing field of work assessing programmatic evaluation of API usability, we continue to explore whether the wisdom of the API design team can be captured in our tooling"

Since Google's initial publication, API governance has only grown in importance. For example, "45% of IT leaders identified API governance as a critical component of their API program"[1] according to a survey conducted by Google Cloud of 770 technology leaders.

In this paper we share an update on the API Governance program at Google, building upon our earlier work on API Design Reviews. Specifically, we introduce three new components, which are utilized in addition to the aforementioned API Design Reviews. 1. API Improvement Proposals (AIPs), a citable single source of truth for API design rules, inspired by the Python Enhancement Proposals (PEP) standards process. 2. An API Linter, that can check and report design violations to API developers the same way program compilers currently report syntax errors or type errors. 3. API Readability, a process to formally educate, certify, and legitamize API Design expertise, providing API Reviewers with additional credibility during API Design Reviews.

The goal of this paper is to present an Action Research style approach to help address the question of how one can build a program to effectively support API Producers in designing high quality APIs. In this paper we a) present and describe the aforementioned three API Governance components; b)Provide the results and discuss the implications of a mixed-methods research study to evaluate the utility of the approach. In the sections that follow we first, provide a recap of our prior work on API Governance, specifically, API Design Reviews [16], and highlight some of the prevailing issues that remained. Then, we introduce the new elements of API Governance, designed to help address these issues, and help meet our ultimate goal of creating a scalable API design program. Finally, we present the results of an evaluation into these new components, and a discussion on the implication of these findings for supporting API producers. Table 1 defines terminology and relationship between terms.

2 Recap of Apiness

In 2012, Google developed the Apiness (API Happiness) program as a way to improve the usability (specifically the quality and consistency) of the APIs they were producing. Macvean et al. shared an overview of the API Design Review piece of the program [16], reflecting on the early success through the lens of the stakeholders having their APIs reviewed. Similar to other API Review processes (e.g., [7]), Google API Design Reviews consisted of independent API Design Reviewers shepherding an API Owner through a series of iterative "heuristic" reviews of their API design, comparing against pre-established Google standards for API design (e.g., naming conventions). Importantly, a number of artifacts, including code samples, sample implementations, etc, were submitted alongside the API specification, as a way to more holistically evaluate the API under review. To train and calibrate new API Design Reviewers, a Shadow Design Review process was implemented. This process allowed new reviewers to observe and participate in API Design Reviews alongside experienced reviewers. As a result, the team of Design Reviewers was able to scale from 3 to 8 in about 10 months. API owners typically requested an API Design Review when they felt that the specification had met most of the product requirements. Once a reviewer was assigned, they would work with the API owner throughout the review process until approval was granted. Future updates to the same API or new versions would typically be assigned to the same reviewer.

Table 1: Table Key API Governance Terminology

Term	Definition
API Governance	A program in an organization that manages API development from design consistency to launch requirements
Apiness	Name of Google's API Governance program
API Design Review	A process where a qualified API design reviewer reviews the API owner's design
API Design Standard	An API design guideline, pattern or building blocks
AIP	API Improvement Proposal - an API design pattern

3 Evolving API Design Reviews

Google presented positive initial feedback on their API Design Review process [16]. Participants in the process, specifically the API Owners having their APIs reviewed, were generally satisfied with the process, and were of the opinion that it had positively impacted the quality of their APIs. API Design Reviewers were valued as not only improving API quality, but also validating and giving confidence to the approach of the API Designers. However, Macvean et al. also discussed a number of areas for improvement in the process, specifically around further scaling the program. This included reflections on how to ensure consistency in feedback (e.g., as the number of active reviewers increased) and how to balance API expertise with product domain expertise (e.g., generalizable API design advice vs product specific concessions), all while ensuring the efficiency of the process was maintained. In the rest of this section we further reflect on a few specific problem areas identified both from research by Macvean et al, and emergent as Google continued to roll out and scale API Design Reviews. Importantly, we highlight the evolution of the API Design Review process to mitigate these concerns.

3.1 APIs to Launch > # of API Reviewers

"*The number of web APIs produced by Google's various business units grew at an astounding rate over the last decade, the result of which*

[1] https://cloud.google.com/blog/products/api-management/7-api-management-use-cases-rising-in-prominence

was a user experience containing wild inconsistencies and usability problems" [16]. Although an anticipated problem, ultimately API Design Reviews could not scale quickly enough to handle the growing number of APIs Google was producing. There were more APIs to launch than reviewers to review them. As a result, the turnaround time from review 'Kick-Off' to a formal sign-off was increasing. In order to maintain the quality and consistency of API reviews, Google had a robust process for qualifying as a reviewer. This includes taking live API design classes, and then shadowing a number of API reviews. API Reviewers were themselves 'volunteers'. E.g., they were predominantly software engineers passionate about API design, rather than hired specifically to do API reviews. As such, all API Reviewers had to balance their time with their 'day job' responsibilities.

3.2 API Reviewer Assignment

Another problem involved the assignment of an API Design Reviewer to a particular API launch request. Upon receiving a review request, a technical program manager would manually assign a design and shadow reviewer using a spreadsheet. This became challenging as the manager had to not only round-robin the reviewers but also keep track of the reviewer's availability. The manager quickly became another bottleneck in this process and scaling became very difficult.

3.3 API Design Review Wall

Another problem was the state at which reviews began. Given the naming ("review") it was quite common for teams to iterate on their API and arrive at a review stage thinking they'd done a good job and would be quickly stamped with an approval. In some cases this was the result. In many others the results varied from minor naming changes all the way to significant structural changes due to oversights by the API-producing team. At that point, the review process acted like a wall, stopping the API launch process in its tracks and forcing the team to start over (or significantly revise) the API in order to climb over the wall. We wanted the system to act more like a ramp than a wall helping teams to start their process of meeting the launch criteria earlier on rather than at the last minute.

3.4 Inconsistent APIs with No Citable Rules

There was inconsistency in reviewer guidance, in part because the API design guidance itself was evolving i.e. not all lessons had been documented. Therefore, API designers sometimes had to create guidance as they went or would give mixed messages. We knew, from previous research [16], that as Google expanded, API Design guidance would be tested in nascent domains (e.g., new product areas, new user archetypes, etc), as such, we would be learning and growing as we went.

As a result of this, Google had API style guides and best practices documented in multiple places, and without the right life cycle management, granularity, or level of detail required to function as high-quality citable materials.

Additionally, inconsistent APIs would cause the iterative review process of reaching a consensus between the API Owner and the reviewers to get lengthy and confusing. This caused many escalations of disputes to higher authorities for resolution as API design review was a requirement for launching a publicly exposed API, and this process blocked the launch increasing overall latency. Achieving API consistency across many product areas and engineering groups could not be scaled with these reviews. We needed a better way to enforce these standards with concrete citable rules.

4 Solving the Problems

Similar to the security concept of defense in depth[2], we introduced a multi-layered process to maintain a high API quality bar while further scaling the API design review process in an efficient manner. We introduced an API governance program with a four-pronged approach that layers and stacks on top of each other:

- **API Improvement Proposals (AIPs)** providing source of truth citable rules.
- An **API linter**, an automated tool that checks APIs for violations of the AIPs giving API developers instant feedback during their development process. By enforcing the design rules, APIs would become more consistent thereby building a ramp to the API Design Review.
- An **API Readability** program that creates a pool of informed and educated API reviewers helping to solve the design review latency and legitimizing the volunteer time donated by API reviewers.
- API design reviews as a **final** large design catch all.

In the rest of this section, we provide a detailed overview of the three new phases of the program.

4.1 Introducing Citable Standardization: AIPs

In 2019, we introduced AIPs (published on aip.dev) to help unify and standardize that guidance in an accessible way.

Inspired by Python Enhancement Proposals (PEP), we took all the existing API style guides and rewrote them into concise rules. The objective was to create a citable and unambiguous style guide that enables engineers to develop consistent, scalable, and flexible API specifications that would not be misinterpreted. AIPs use the following requirement level keywords: "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY", which are to be interpreted as described in RFC 2119 [3].

We found that in order to fully internalize and understand the guidelines, API Designers needed to 'see them in action'. Therefore many of the AIPs contain concrete examples to solidify standards. For example, the pagination AIP (AIP-158), shows an example of the pagination design guidelines in action using a book analogy (see figure 1).

There's a few different stakeholders involved in the AIP process (as shown in figure 2). For an AIP to get final approval, either the design or infrastructure TL responsible over the domain covered must approve.

- **API Producer:** The API producer that is proposing a new AIP or change to an existing AIP.
- **AIP Editor:** The editors are responsible for approving AIPs and for the administrative aspects of shepherding AIPs and managing the AIP pipeline and workflow.

[2]https://en.wikipedia.org/wiki/Defense_in_depth_(computing)
[3]https://www.ietf.org/rfc/rfc2119.txt

```
// The request structure for listing books.
message ListBooksRequest {
  // The parent, which owns this collection of books.
  // Format: publishers/{publisher}
  string parent = 1 [
    (google.api.field_behavior) = REQUIRED,
    (google.api.resource_reference) = {
      child_type: "library.googleapis.com/Book"
    }];

  // The maximum number of books to return. The service may return fewer than
  // this value.
  // If unspecified, at most 50 books will be returned.
  // The maximum value is 1000; values above 1000 will be coerced to 1000.
  int32 page_size = 2;

  // A page token, received from a previous `ListBooks` call.
  // Provide this to retrieve the subsequent page.
  //
  // When paginating, all other parameters provided to `ListBooks` must match
  // the call that provided the page token.
  string page_token = 3;
}

// The response structure from listing books.
message ListBooksResponse {
  // The books from the specified publisher.
  repeated Book books = 1;

  // A token that can be sent as `page_token` to retrieve the next page.
  // If this field is omitted, there are no subsequent pages.
  string next_page_token = 2;
}
```

Figure 1: Code example from AIP-158 about pagination.

- **Infrastructure TL:** The infrastructure tech lead that will resolve final infrastructure escalation such as security, logging issues or regulatory compliance.
- **Design TL:** The tech leader that will resolve final design escalations.
- **TL:** The technical lead who's the final decision-maker on the AIP process and the final point of escalation if necessary.

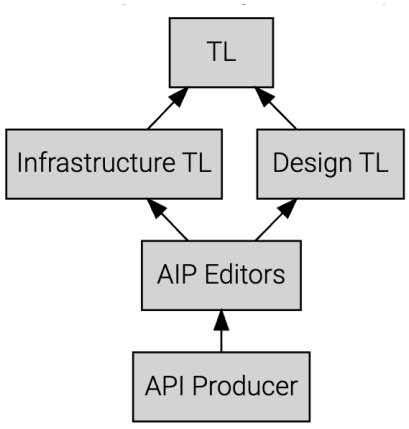


Figure 2: The stakeholders of the AIP process.

Another objective was to centralize the API standards so that the information architecture was consistent and predictable. This allowed engineers to easily get to an AIP, for example aip.dev/122, and also allowed collaboration through standard channels (e.g., submitting change lists). Additionally, each AIP is assigned a specific 'state' and can only be in a single state at any given time (see life cycle in figure 3):

- **Draft:** The initial state for an AIP which means that the AIP is being discussed and iterated upon, primarily by the original authors.
- **Reviewing:** This is the next stage which means that the authors have reached a general consensus on the proposal and the editors are now involved.
- **Approved:** This state means that everyone agrees upon the proposal and is considered "best current practice".
- **Withdrawn:** The state when the author withdraws their proposal.
- **Rejected:** This state means that the AIP editors rejected the proposal. They remain as rejeccted to provide documentation and reference to inform future discussions.
- **Deferred:** This state means that the AIP has not been acted upon for a significant period of time.
- **Replaced:** This state means that the AIP has been replaced by another AIP.

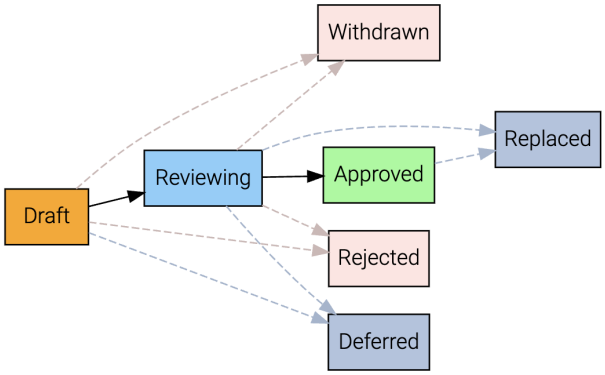


Figure 3: The life-cycle of an AIP.

4.2 Introducing Early Enforcement: Linter

How do we empower and train API Owners to design better APIs that are consistent and improve design review latency? How do we build the ramp that we discussed in section 3.3 so API Owners don't hit a wall at the Design Review, but rather, receive consistent and iterative feedback? With scale in mind, we turned to software to help with code enforcement, adopting tools and processes already familiar to software developers.

Thus we introduced the API linter (linter.aip.dev) to help "programmatize" elements of the review process, getting best practices to the API Owner during development. Linters are static analysis tools that are a popular way to maintain code quality and enforce standards by flagging bugs or incongruous code. Prior research has been conducted on linters, for example Kavaler et al. explores their usage in JavaScript projects [11], including validating their use in multiple different domains such as dependency management, game development [3], web development [14], mobile development [13] [9], gui animation [28], security [8], sql [21] and others [27].

Our API linter is written in Go and can be installed with a simple "go install github.com/googleapis/api-linter/cmd/api-linter@latest." At the time of writing, approximately two thirds of existing AIPs have been codified in the linter.

Protos, or Protocol Buffers, are Google's way of serializing structured data in a language/platform neutral way. Our API linter started as a closed-source linter and ran on all non-private protos inside Google's code base. It's the first line of defense when anyone checks in code well before it ever hits an API reviewer. The linter alerts the engineers when they are doing something inconsistent with an AIP Rule and redirects them directly to the appropriate AIP. It can be run manually in the terminal, but developers using Google's code editor, Cloud Integrated Development Environment and Repository (CIDER), get immediate feedback as they type. For example, the AIPs provide guidance that "When defining a resource, the first field should be the resource name, which must be of type string and must be called name for the resource name." So if we have a resource called *'Foo'* and it has a missing *name* field, when the API owner runs "*api-linter proto_file*", the linter would output *"Message 'Foo' has no 'name' field,"* and point to the appropriate AIP without requiring a reviewer as shown below:

```
problems:
  - message: Message 'Foo' has no 'name' field.
    location:
      start_position:
        line_number: 5
        column_number: 1
      end_position:
        line_number: 10
        column_number: 1
    rule_id: core::0123::resource-name-field
    rule_doc_uri: https://linter.aip.dev/123/resource-
    name-field
```

So, instead of receiving potentially disruptive guidance late in the design process, the linter guides by giving small incremental improvements and feedback along. Additionally, by meeting API Designers 'where they are', i.e., during development, in the context of API design, the linter aids in the discoverability and consumability of AIP guidance.

The linter acts as an enforcer of the AIPs and allows minor design flaws to be resolved by the engineer even if they don't know the standards. Additionally, by allowing 'smaller' guidance to be provided in an interactive fashion, it allows the API Design Review process to be more on the larger, more novel API design decisions. I.e., during the API Design Review itself, API Design Reviewers are no longer 'bogged down' by the smaller more routine occurrences.

The overall API design review process including AIPs, linter and API design review is shown in figure 4.

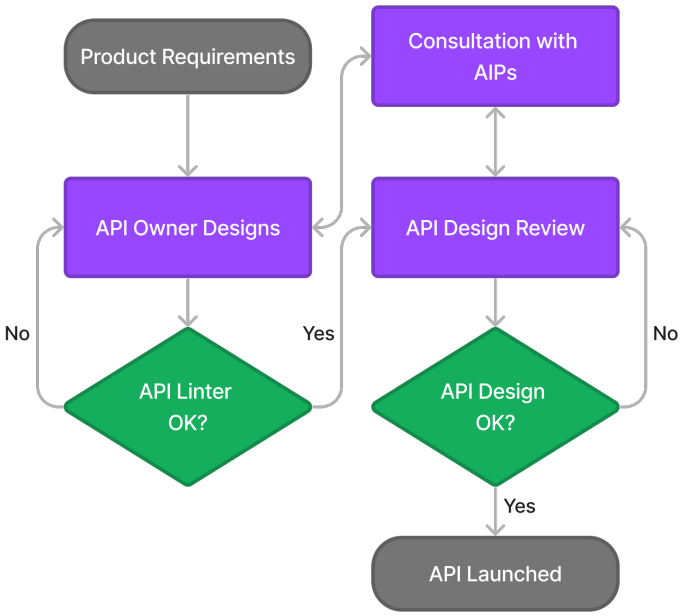


Figure 4: The full API design process from requirements to launch.

4.3 Introducing API Readability

Google has a "company-wide readability training process, whereby experienced engineers with a passion for code readability can opt in to become certified, then train other engineers in how to write readable, idiomatic code in a particular language. They achieve readability by first demonstrating their own code standards, with an existing readability reviewer auditing a substantial change or series of changes until the reviewer is satisfied that the author knows how to write readable code in a particular language"[10]. Readability in a language grants special disposition, for example, anyone can review Python code but only a reviewer with "Python Readability" can approve the code change to be submitted into the code base. In prior research Aggarwal et al. outlined that source code readability is crucial for project maintainability [1] while other research have claimed that readability of code should be part of the development phase [6] [12].

Thus, In 2018, to address one of the problems of needing to increase the qualified reviewer pool (as mentioned in section 3.1), we introduced "API Readability." API Readability was a stamp of approval that a person is qualified and able to review the design specifications of an API, and that they have gone through specialist training to achieve this domain specific knowledge and expertise. API Readability was in part achieved by passing a self-service exam based on newly launched AIPs. Because much of the API Design expertise had been captured in the AIPs, the goal of the certification was as much about teaching reviewers to know where to look for the guidance, versus having them memorize the guidance itself. We wanted more people saying *"I think there might be a rule about this on AIP.dev."* or *"I think we should ask API reviewers mailing list because this is a tricky issue that likely has never been covered."*

Because Readability was a familiar and well established process at Google, it provided an incentive for potential API Reviewers to become more involved in API Governance. It added a degree of credibility, which helped them justify their involvement (i.e., their volunteer time doing API Design reviewing became legitimized as a more formal process). In turn, even though the process of becoming an API Reviewer was evolved to contain 'more hurdles', the pool of reviewers still increased.

4.4 Automate assignment of reviewers

Google API Owners submit their design specifications, defined in protocol buffers (protobufs), using an internal code review tool called Critique. Then they add API design and shadow reviewers in the tool that was assigned to them manually by the technical program manager. This quickly became a laborious and unscalable process as mentioned in section 3.2.

Since API reviewers with API Readability were now members of a private Google group, we were able to leverage an internal plugin to automatically (in a round robin fashion) assign reviewers from the group to the code review. Additionally, this tool automatically considered reviewers that were available (excluded folks that were out of office). This automation scaled the assignment process by taking the program manager out of the loop.

5 Evaluation

As previously described, prior research [16], outlined the potential of an API Design Review program, while also providing scope for further refinement and future evolution. In Section 3 and 4, we highlighted some of the top opportunities and the new features designed to address them. In this section we outline an initial evaluation of these features, and the extent to which they come together to provide our ultimate goal of creating a scalable API Governance program capable of guiding the development of consistent and usable APIs.

5.1 Overview and Goals

To this end, the ultimate success of an API Governance program can be summarized as follows:

(1) Positive perception from API Producers. To what extent do API Producers believe that API Governance is helping them in efficiently building high quality APIs.
(2) Improved Quality of APIs. To what extent does going through an API Governance process actually result in higher quality APIs, both on an individual per API basis, and when viewed collectively

In this research, we focus primarily on the first goal, perceptions of API Producers, with some early insight into the second goal. Future Research, outlined in Section 8, will focus on fully validating the impact of API Governance on API Usability.

In this paper we focus on the following four Research Questions:

- **RQ1** What is the overall sentiment of API Producers towards the new API Governance processes, specifically, AIPs, API Linter, and API Readability.
  - **RQ1.1** What is the overall sentiment towards the API Governance process?
  - **RQ1.2** What is the overall sentiment towards AIPs?
  - **RQ1.3** What is the overall sentiment towards API Linter?
  - **RQ1.4** What is the overall sentiment towards API Readability?
- **RQ2** What are the primary painpoints in API Governance as it exists today?
- **RQ3** To what extent do API Governance processes help or hinder the API design process?
- **RQ4** When considering the API Governance processes, what are the primary user groups who stand to gain the most from them?

5.2 Methodology

In order to begin answering the aforementioned research questions we conducted a two-phased, mixed-methods, research approach. The first phase consisted of a large-scale survey, sent to participants of an API Interest group. The survey contained a number of likert-style questions designed to measure perceptions towards API Design and API Governance specifically. The survey was designed to give us a broad overview of sentiment towards API Governance, while providing some initial qualitative 'themes' to go deeper into during our second phase. The second phase was an interview study conducted with a subset of the Phase 1 survey participants. The

interviews probed more deeply into the participants' experiences of API Governance.

5.2.1 Phase 1. Large Scale Survey. We sent the survey to the Google API interest group, which contains approximately 4000 members. This is a self opt-in mailing list, where members are anyone with interest in any aspect of APIs (e.g., API Usability, API Security, Designing APIs and Frameworks, etc). Our estimation is that it contains a considerable proportion of those actively designing APIs at Google. Of all who received the email, we requested responses from only those who have completed an API Design Review in the last 2 years, ensuring that respondents will have interacted with the API Governance features (outlined in RQ1). In total, we received 125 responses to the survey.

Participants were sent a survey containing a mixture of likert style questions (e.g., Generally speaking, to what extent do you agree with the following statement: AIPs help me design better APIs) augmented by open-ended questions to elicit qualitative experience feedback (e.g., You said frustrating to the prior question, what makes the API design review frustrating?). The survey also asked participants to self report their level of experience designing APIs (e.g., length of time, number of APIs) and experience working with API Governance (e.g., level of familiarity with API Governance, whether they have achieved API Readability, etc).

5.2.2 Phase 2. Deep Dive Interviews. Participants in our Phase 1 survey were able to opt-in to participate in future phases of the research. For our Phase 2 interviews, we invited a random subset from that group. Interviews were semi-structured in nature, designed to explore more deeply the aforementioned RQs. We asked participants for more information on their background, then to provide greater context on the short-form responses they had provided during the survey. For example, "You stated you have API Readability, what was your primary motivation for gaining this?". In total, interviews lasted between 30 minutes and 45 minutes.

5.2.3 Analysis. For our survey, we computed descriptive statistics for all of the likert-style questions. Kendall rank test was used to evaluate the correlation between survey responses. To prevent false discoveries, the Benjamini-Hochberg procedure was used to calculate the new significance level. We set the False Discovery Rate at 0.1 to avoid missing out important correlations, resulting in a significance level of 0.0456. All hypothesis tests were evaluated using this significance level. All qualitative responses were coded using a lightweight content analysis by the two primary authors.

For our interviews, all sessions were transcribed and then coded using the same lightweight content analysis process used for our survey.

In the sections which follow we integrate the results from both Phases of our research in order to provide a more comprehensive answer to our Research Questions. All numerical data (e.g., descriptive statistics, correlations, etc) are based only on the survey responses. Where Participants are discussed (e.g."P2 stated") these quotes were based on our interviews.

5.2.4 Participants The API design experience of the participants ranged from 1 to 20 years ($\overline{x}$ = 5.8, σ = 4.4, median = 5) and with the range of completed API Design Reviews (and thus participation in API Governance) ranging from once to over 12 ($\overline{x}$ = 3.9, σ = 4.04, median = 2). Roughly 70% of the respondents have API Readability with another 12% in the process of completing it.

5.3 Results

5.3.1 Overall Sentiments

5.3.1.1 RQ1.1 What is the Overall sentiment towards the API Governance process? Overall, sentiment towards the API Governance process was largely positive, as reflected in both survey responses and interviews. Participants consistently reported that API Governance improved both the quality of the outcome (highlighting consistency, usability, and maintainability as the primary drivers) and process of building APIs. For example, P6 stated "Strongly believes it has and will improve API quality. If you don't have a framework to guide people how to write APIs even small components of services, for example different requests within RPC methods, can be written in different ways and that can confuse customers". While, P8 highlighted the API Governance process as being a key learning feedback loop "I think of Governance being a feedback loop of learning and should teach the people who engage with it to learn how to meet a certain quality bar"

Given past research on API Design processes had highlighted speed as a potential source of concern [16], We asked participants to rate their agreement with the phrase "the pace of the API design review felt appropriate". 59.2% reported the API design review pace appropriate, 32% neutral and 8.8% found it slow. We also gathered data on how hard the process was to adhere to, and level of frustration with the process. 10.2% of the respondents reported that they found the process frustrating, while 24.4% reported that they had to work hard to get their API through the process. There was a statistically significant (p-value < 0.001) negative correlation (τ = -0.43) between pace and finding the process frustrating.

For newer users navigating the process for the first time, our qualitative data highlighted that it could be slow, as one learns to navigate through the recommendations of AIPs. Conversely, experienced practitioners argued there was a speed benefit, e.g., P3 stated " I don't need to spend time arguing with others on why we need to design it this way. Saves us a lot of time to point people to AIPs?"

Given the complexity of designing APIs, and the hope the codified guidance in the form of AIPs and automated linters would alleviate at least part of this complexity, we were also interested in understanding how mentally taxing the API producers found the process to be. Overall, 37.8% of respondents rated the process as "mentally demanding", with 45.7% reporting neutral levels, and the remaining 16.5% reporting little levels of demand.

In the sections that follow, we go into more depth on the specific elements of the program.

5.3.1.2 RQ1.2 What is the overall sentiment towards AIPs? 84.3% of survey respondents were of the opinion that AIPs were able to answer "most of their questions when designing APIs", with 89.0% agreeing with the statement that they helped them design better APIs and 65% reporting that AIPs help them design APIs faster. P5 stated " it's really good and helpful when there's a debate as we can point people to the [AIP]"

While positive towards AIPs, P7 recommended extending the AIPs to more clearly articulate the background and rationale for a decision. "Motivation behind the recommendation. Usually people just comply however it would be easier to remember if you knew the motivation behind it. Your mental model will map to it better so you will remember it better.", highlighting not just their desire to remember and adhere to the rules, but use AIPs as a source for learning.

One clear area of improvement for AIPs is around community engagement. One goal of AIPs is that they grow and evolve to the needs of the users. Although there is a tension here (e.g, ever evolving best practices can be confusing to those that regularly design APIs), when there are gaps not covered by AIPs, we hope that API Producers can highlight and help close them. Only 26.8% of survey respondents felt empowered to contribute to AIPs, with 16.5% agreeing that it is easy to propose changes.

5.3.1.3 RQ1.3 What is the overall sentiment towards API Linter? 81% of the survey respondents had interacted with the linter in various degrees and, among them, 80% (meaning 65% overall) found that the linter helped them design better APIs. Participants in our interview reported that the linter not only helped 'enforce the rules' of AIPs (e.g., ensuring adherence to the guidance in a consistent manner), but also impacted the way in which they approached API Design. E.g., P2 reported using the linter as a pseudo reviewer / thought partner, providing the first round of feedback before consultations with other engineers on their team. Having access to the linter also led to more iterative prototype oriented development, according to P4, who appreciated the tighter feedback loops as they designed their API. The linter was also perceived as being a critical learning tool, helping to educate API best practices.

23.6% of survey respondents reported having to override the linter guidance, highlighting one potential area for future improvement. Our survey did not explicitly ask for further details why, and none of our interview participants had feedback on this topic, highlighting an area for future research.

There was a positive correlation (p-value = 0.003, τ = 0.21) between the number of API reviews completed as a producer and whether they find the linter helps them design better APIs.

5.3.1.4 RQ1.4 What is the overall sentiment towards API Readability? Consistent with the other elements of API Governance, API Readability was viewed positively from respondents, with 91.3% of survey respondents agreeing that it is important for API consistency, and 87.4% agreeing that it meaningfully improves the quality of APIs.

There is statistically significant (p-value = 0.02) positive correlation (τ = 0.16) between the number of API reviews completed as a producer and whether or not they find API Readability worth getting.

5.4 Summary

In summary, each of the newly introduced elements of API governance were met, both in isolation and in sum, with largely positive perceptions from the study participants. In all of the cases, there remains room for continued improvements, which we unpack in further details in answer to RQ2.

5.4.1 Areas for Improvement. RQ2. What are the primary painpoints in API Governance as it exists today? While the above section highlights the general positive sentiment towards the process, we also observe a number of noticeable areas for further improvement.

With only 59.2% agreeing that the pace of the entire process is appropriate, we must continue to evaluate ways to support the complex and demanding task of designing and building APIs in an efficient way.

Two general themes emerged from the open ended feedback, both relating to the overall pace / level of frustration involved in the process. 1. Occasional Inconsistency in guidance. 2. A lack of guidance on more nuanced and complex use cases.

For example, while one respondent valued the expertise of API reviewers, they noted "*Sometimes there is inconsistency of views among reviewers*", something which can slow down the process, and undermine trust in the expertise. Another respondent noted "*Adding some more complicated use cases to the docs page could be helpful.*"

P8 also provided context in our follow-up interviews. "*For 75% of API work, it reduces the thoughts substantially... Follow AIP, Linter fixes and I'm done. The remaining 25% when things don't fit a pattern: If there's a tricky API, I'm gonna page around the AIPs to see if there's a pattern that can be leveraged. If none of that helps, then there might be direct conflict.*". We revisit this discussion on edge cases, and consistency for complex APIs in Section 6.3.

P4 had a nuanced perspective on the speed of the process, with how you view API Governance as a key differentiating factor in your experience. "*Some people see a lot of red tape in having to jump through the hoops and don't engage as deeply as I hoped they would: "Let's just make the linter happy or make the API reviewer happy"... It's a reminder to think about the big picture*"

Further to this, P6 described the need to better evangelize and educate users on API Governance. "There's people aware of AIPs and some that are not aware. Even within people that are aware, some follow it strictly and others do not."

Finally, although improvements to the lifecycle management of AIPs was recognized, participants also noted that because so much active API Design decisions and discussions were occurring, and the software industry was so fast moving, it was occasionally the case that best practices were not yet codified back into the AIPs efficiently enough.

5.4.2 Impact on API Design. RQ3. To what extent do API Governance processes help or hinder the API design process? In the preceding sections we have highlighted overall positive sentiment towards API Governance, and its role on API Design. Our linters and readability process, for example, are perceived as aiding in the quality and consistency of the APIs they help produce. P9 highlighted both the engineering benefit (e.g., improved API usability) and business benefit (e.g., "reduced cost of building APIs"). As previously discussed, not only is there a perception that API Governance improves the quality of the APIs, it has also impacted the API Design process itself, helping API producers get feedback more efficiently and iteratively. Our respondents also recognize the continued improvement of the process over time, for example, "*API review process has come a long way with improvements over the years!*"

5.4.3 Benefit By User Segment. RQ4. When considering the API Governance processes, what are the primary user groups who stand to gain the most from them? We analyzed the extent to which API Design experience (measured via self-reported years of experience developing APIs professionally) correlated with perceptions towards the elements of API Governance. We found that there was a statistically significant (p-value = 0.04) negative correlation (τ = -0.12) between level of experience and finding the Governance process frustrating.

This trend was observed throughout the elements of API Governance, for example, those with more experience of using the linter (p-value < 0.001) positive correlation (τ = 0.41) AIPs (p-value = 0.01) positive correlation (τ = 0.17), and API Readability (p-value = 0.012) positive correlation (τ = 0.16) found them to have a higher positive impact on the quality of API produced.

On the one hand, this is somewhat logical in the sense that those with higher levels of API expertise have likely had more first hand experience on the difficulties associated with effective API design, and thus the need for API Governance processes to exist. On the other hand, one would expect these tools to provide more direct support for less tenured API designers. Indeed, P8 highlighted the benefits of API Governance as a 'learning tool'. If we consider AIPs for example, one may imagine that the more APIs you've designed, the more you would a) Have internalized the guidance, and have less need to consult AIPs; or b) Have well set biases which conflict with AIPs, and thus, find them more obstructive to your own beliefs of 'good' API Design. We discuss more on the implications of this in the onboarding section 6.2 below.

6 Discussion

6.1 The value of API Governance

Our participants generally agree that API Governance is necessary in enforcing API consistency in large organizations, and when done right, that it is a tool in aiding in the process of designing and developing usable APIs. Since governance means extra process, it must be approached in an iterative fashion with incremental improvements. For example, although 59.2% of our survey participants reported that the design review process is appropriately paced, there's still room for improvement.

Our respondents see the value of having AIPs as a single source of documented truth, providing a citable resource that can be used during the design process to aid in design discussions, and codified into tooling in order to provide iterative and in context feedback in a timely fashion.

Our results suggest that the biggest time issues come where there are nuanced and unresolved API design decisions which have not been covered in existing AIPs, i.e., the "The remaining 25% when things don't fit a pattern" that P8 previously described. In such cases, API design experts must use their collective wisdom, and reach consensus in a way that scales to the needs beyond the immediate API. Given the fast moving and ever evolving state of the software industry, the goal will never be to have 100% of all possible design decisions captured in a rule. The fact that (in the eyes of P8) 75% have been captured represents considerable success. Scaling the pool of reviewers, codifying rules into a single source of truth, and baking those rules into approachable tooling has helped improve the API design process for the majority of cases.

6.2 Onboarding New Users to API Governance

Generally, the more often participants go through the governance process (including using the AIPs and API Linter), the more value they see and the less frustrating they find it. Our research suggests there are a few possible reasons for this including a) Familiarity with API governance (e.g., expectations have been set, tooling such as the linter has been set up and configured, etc) and b) Appreciation of the need for Governance, e.g., seeing the bigger picture challenges of designing usable and useful APIs in a consistent way.

This does however lead to a new research problem, how do we make the process more valuable and easier for new users, too.

Certainly, we must continue evaluating the usability of the tools we provide, to reduce friction in the process of adopting and utilizing them. For example, our research did highlight scope to improve the usability of AIP.Dev (e.g., improve the search functionality to make it easier to discover rules) and the linter (e.g., improve the installation experience).

However, there is also scope to do more to ensure newer users to API Governance see the same value as their more tenured colleagues. Given API Design is recognized as a specialist and difficult skill, with numerous challenging facets [17], there may be opportunity to more formally lean into Governance as not only a tool to ensure adherence to rules, but like our participants reported in our study, utilize it as a formal teaching and learning mechanism. This happens implicitly in part today, but as we continue iterating on the program, we wish to explore ways to evangelize the importance of API Governance and API Design best practices, while also exploring ways to address some of the common API Design problems identified as challenging in past research [17].

6.3 Consistency and Complexity

Although AIPs have made considerable progress in ensuring there is a single source of truth for API Design guidelines, and thus, eradicating some of the previously described issues around diverging or inconsistent guidance, there still remains scope for continued investigation around this topic.

We believe, as we continue to codify and publish best practices into AIPs, and translate them into our other tooling such as the linter, we can create an effective AIP Governance program for the vast majority of APIs. Our participants agree. However, as previously discussed, the ever changing and fast paced nature of the software industry means that a)There is a lag in ensuring best practices are fully covered in our official documentation; b) There will always be nascent and emergent areas that require deeper and more nuanced design discussion. Thus, once could frame the goals of API Governance as follows. 1. For the 'standard' API, for which AIPs do exist, ensure a process that makes that information as accessible and consumable as possible, such that these APIs can be designed in an as efficient way as possible. 2. For APIs that have nascent and nuanced design decisions, create an environment that ensures meaningful discussions (e.g., bring together qualified experts with knowledge of standards, understand the trade-offs and implications of design decisions, etc) in an efficient way as possible.

While we believe we are achieving much of that with our program, one way to further scale against both of these goals is by better federating and democratizing. As mentioned earlier, only

27% of the developers feel empowered to propose changes to AIPs, so the best remediation is to encourage and empower developers to be more involved in submitting PRs to change AIPs. As previously discussed, many users frequently override warnings from the linter. There may be scope to capture in the moment why this is happening (e.g., is the rule not applicable in this context?) and create a positive feedback loop which updates the best practices.

6.4 Future of API Governance

The new process still involves a lot of human intervention. The manual labor in conducting individual API design reviews are fixed. We need to switch the labor from the reviewers to the producers because the number of reviewers are fixed while the producers scale. But of course, the reviewers have the expertise and the producers don't. The API linter catches most minor design issues however complex APIs still require human reviews. We need to find automated systems of transmitting expertise from reviewers to producers to solve complex use cases which may involve updating the AIPs themselves.

Our layered approach has augmented the API design review approach. However, future work should move towards full automation while reducing the burden on people. We need to analyze why APIs are being reviewed and then pick out pieces that can be automated. AI can highlight APIs that need special attention and therefore make API design reviews an exception. Ideally, we will be able to scale the API design review process with a small set of reviewers. This could scale because the exceptions that aren't covered by the linter or documentation would become fewer and fewer. So as the number of reviews increases, the number of exceptions that need special handling decreases.

The final step to achieve full automation is to leverage large language models (LLMs), such as ChatGPT, to automatically code the API specifications based on a list of requirements. Recent research such as the work from White et al. [26] shows the potential of LLMs to produce API specification however there's still a significant amount of human involvement and expertise "*currently necessary to leverage LLMs effectively for automating common software engineering tasks*"[26]. While prompt patterns can help mitigate issues such as incorrect code, a key prerequisite to leveraging AI is to have a strict set of design principles which leads us back to AIPs. Since most organizations have nuanced differences in API design principles, the prerequisites for leveraging AI will require all organizations to first create a comprehensive list of codified standards, such AIPs, to handle most use cases. Therefore, we believe further investment in refining AIPs will make the future transition to AI generated API design specifications much more feasible.

7 Threats to Validity

Single organization. The API Governance process, AIPs and Linter was specifically designed for a single organization. Depending on their size and tooling resources, other organizations may face different challenges. Our hope is that since the process and tools introduced is enabling a large tech organization to achieving API consistency, it will inspire and/or influence other large organizations to do the same.

Selection bias. Those surveyed cared enough to be in the Google's API interest group. There could have also been some sort of social desirability bias, e.g., participants are more inclined to be positive given the process came from their employer. Out of 4000 members in the Google API interest group, only 125 (3%) participated in our study. Below are some potential reasons behind the low participation rate from the interest group members:

- members has other API interests (not API Design Reviews)
- the last API design review completed by the member was more than 2 years ago or the member has never completed an API design review
- member is an inactive employee of Google

Lastly, participants who already had API Readability, could be biased in classifying API design reviews as effective.

8 Next Steps

Although sentiment was largely positive, we will continue iterating on API Governance in order to build an optimal program. Qualitative feedback shows that future work is needed in bridging consistency among API Design Reviewers and introducing complex use cases in the AIPs. Additionally, even though AIPs are concrete citable rules, additional research is needed to quantify if AIPs and/or the API Linter are in fact producing better APIs both from a production and consumption perspective. To be able to answer the question "Do AIPs and/or the API Linter really make APIs better?" Can we find a way to define API goodness? Further research is also recommended in analyzing the effectiveness of individual AIPs (for example AIP-155 resource identification or AIP-164 soft delete).

9 Conclusion

In this paper we introduced API Improvement Proposals (AIPs), API linter and API Readability as three core facets of API Governance, designed to build upon existing API Design reviews. Together, these were designed to create a process which supported API producers in building usable APIs in an efficient manner.

Our research shows that sentiment towards AIPs, the API linter and API Readability are generally positive, validating the general direction of API governance in supporting API producers. We believe the results of this research can be of benefit to any organization that produces APIs, by highlighting different techniques which can support API producers in undertaking notoriously difficult tasks, in a way that introduces friction "in the right ways". The tools described in this paper, including the API linter and AIPs, are publicaly available for use today.

We also present a number of avenues for future research directions, for both better understand and improving the API design process. We hope that companies, and individual API designers, find this information useful in order to help improve their own API design processes.

10 Acknowledgments

Thank you Luke Sneeringer, Alfred Fuller, Rohith Amanaganti, Richard Frankel, Navid Borjian and Lea Verou for your assistance in this research and preparation of the work.

References

[1] K.K. Aggarwal, Y. Singh, and J.K. Chhabra. 2002. An integrated measure of software maintainability. In *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318)*. 235–241. https://doi.org/10.1109/RAMS.2002.981648

[2] Joshua Bloch. 2006. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. 506–507.

[3] Antonio Borrelli, Vittoria Nardone, Giuseppe A Di Lucca, Gerardo Canfora, and Massimiliano Di Penta. 2020. Detecting video game-specific bad smells in unity projects. In *Proceedings of the 17th international conference on mining software repositories*. 198–208.

[4] Steven Clarke. 2005. Describing and measuring API usability with the cognitive dimensions. In *Cognitive Dimensions of Notations 10th Anniversary Workshop*, Vol. 16. Citeseer.

[5] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 73–84.

[6] James L Elshoff and Michael Marcotty. 1982. Improving computer program readability to aid modification. *Commun. ACM* 25, 8 (1982), 512–521.

[7] Umer Farooq, Leon Welicki, and Dieter Zirkler. 2010. API usability peer reviews: a method for evaluating the usability of application programming interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2327–2336.

[8] Pascal Gadient, Mohammad Ghafari, Patrick Frischknecht, and Oscar Nierstrasz. 2019. Security code smells in Android ICC. *Empirical software engineering* 24, 5 (2019), 3046–3076.

[9] Sarra Habchi, Geoffrey Hecht, Romain Rouvoy, and Naouel Moha. 2017. Code smells in ios apps: How do they compare to android?. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 110–121.

[10] Fergus Henderson. 2020. Software Engineering at Google. arXiv:1702.01715 [cs.SE]

[11] David Kavaler, Asher Trockman, Bogdan Vasilescu, and Vladimir Filkov. 2019. Tool choice matters: JavaScript quality assurance tools and usage outcomes in GitHub projects. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 476–487.

[12] John C Knight and E Ann Myers. 1991. Phased inspections and their implementation. *ACM SIGSOFT Software Engineering Notes* 16, 3 (1991), 29–35.

[13] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. 2017. Static analysis of android apps: A systematic literature review. *Information and Software Technology* 88 (2017), 67–95.

[14] Suryadiputra Liawatimena, Harco Leslie Hendric Spits Warnars, Agung Trisetyarso, Edi Abdurahman, Benfano Soewito, Antoni Wibowo, Ford Lumban Gaol, and Bahtiar Saleh Abbas. 2018. Django web framework software metrics measurement using radon and pylint. In *2018 Indonesian Association for Pattern Recognition International Conference (INAPR)*. IEEE, 218–222.

[15] Andrew Macvean, John Daughtry, Luke Church, and Craig Citro. 2016. API Usability at Scale. In *Proceedings of the 26th annual workshop of the Psychology of Programming Interest Group*.

[16] Andrew Macvean, Martin Maly, and John Daughtry. 2016. API Design Reviews at Scale. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems* (San Jose, California, USA) *(CHI EA '16)*. Association for Computing Machinery, New York, NY, USA, 849–858. https://doi.org/10.1145/2851581.2851602

[17] Lauren Murphy, Tosin Alliyu, Andrew Macvean, Mary Beth Kery, and Brad A Myers. 2017. Preliminary analysis of REST API style guidelines. *Ann Arbor* 1001 (2017), 48109.

[18] Lauren Murphy, Mary Beth Kery, Oluwatosin Alliyu, Andrew Macvean, and Brad A Myers. 2018. API designers in the field: Design practices and challenges for creating usable APIs. In *2018 ieee symposium on visual languages and human-centric computing (vl/hcc)*. IEEE, 249–258.

[19] Brad A Myers and Jeffrey Stylos. 2016. Improving API usability. *Commun. ACM* 59, 6 (2016), 62–69.

[20] Sarah Nadi, Stefan Krüger, Mira Mezini, and Eric Bodden. 2016. Jumping through hoops: Why do Java developers struggle with cryptography APIs?. In *Proceedings of the 38th International Conference on Software Engineering*. 935–946.

[21] Csaba Nagy and Anthony Cleve. 2018. SQLInspect: A static analyzer to inspect database usage in Java applications. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 93–96.

[22] Martin P Robillard and Robert DeLine. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (2011), 703–732.

[23] Christopher Scaffidi. 2006. Why are APIs difficult to learn and use? *XRDS: Crossroads, The ACM Magazine for Students* 12, 4 (2006), 4–4.

[24] Jeffrey Stylos and Brad Myers. 2007. Mapping the space of API design decisions. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*. IEEE, 50–60.

[25] Jeffrey Stylos and Brad A Myers. 2008. The implications of method placement on API learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 105–112.

[26] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. arXiv:2303.07839 [cs.SE]

[27] Zhen Yu, Xiaohong Su, and P Ma. 2016. Mocklinter: Linting mutual exclusive deadlocks with lock allocation graphs. *International Journal of Hybrid Information Technology* 9, 3 (2016), 355–374.

[28] Dehai Zhao, Zhenchang Xing, Chunyang Chen, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Seenomaly: Vision-based linting of gui animation effects against design-don't guidelines. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1286–1297.