API Design Reviews at Scale

Andrew Macvean
Google Inc.
amacvean@google.com

Martin Maly
Google Inc.
mmaly@google.com

John Daughtry
Google Inc.
daughtry@google.com

Abstract

The number of APIs produced by Google's various business units grew at an astounding rate over the last decade, the result of which was a user experience containing wild inconsistencies and usability problems. There was no single issue that dominated the usability problems; rather, users suffered a death from a thousand papercuts. A lightweight, scalable, distributed design review process was put into place that has improved our APIs and the efficacy of our many API designers. Challenges remain, but the API design reviews at scale program has started successfully.

Author Keywords

API Usability; API Design Review; Heuristic Evaluation

ACM Classification Keywords

H.5.3 [Group and Organization Interfaces]: Evaluation / methodology.

Introduction

Peer reviews are a long-used technique for identifying defects in code. They can take many forms, from periodic code inspections [6] to constant peer review in the form of pair programming [2]. Research in this area spans from identifying the characteristic that make good reviews (e.g., [4]) to the social aspects that make it work (e.g., [12]). The focal point, however, has been on the adoption of review practices and the efficacy of reviews towards reducing technical defects, as opposed to reducing usability defects.

This is a problem because developers are users too [3]. When code is written, it is exposed to other programmers by what is commonly referred to as an Application Programming Interface (API). More akin to a command-line interface than a graphical user interface, APIs are how blocks of code turn into systems, and (via web APIs) how systems turn into global software ecosystems.

Unlike a graphical user interface, the use of an API is executed time and time again by a computer. Thus, changes to the shape of the API are more difficult; if you have 50 million users and you do something as simple as fixing a spelling mistake on a button (e.g., subbmit to submit), it is unlikely to cause any problems. If, however, you made the same change in an API, every system that uses your API will fail unless they can be allowed to keep using the old API or developers go in and update their code.

There are a number of decisions API producers must make to construct an API [11], each of which can have a significant impact on the API's usability. Prior research has looked at the usability implications of particular design decisions, for example, whether or not to use parameterless default constructors or explicitly required constructors [10]. There is also a growing field of research on how to test and evaluate the usability of an API, and the applicability of 'traditional' GUI based testing practice. Empirical research has looked at the use of heuristic evaluation (e.g., [8]), lab-based usability testing (e.g., [5]) and generating methods / metrics for programmatically calculating usability (e.g., [9])

Farooq et al. [7] used peer reviews to explicitly uncover usability defects in APIs. As they explain, lab studies for APIs don't scale well, due to recruiting challenges and time constraints. To scale out their API usability efforts, they utilized usability-focused peer reviews. They found the peer reviews to be more efficient than API usability tests, while also effectively highlighting usability issues in the API design. Thus, providing a more scalable method for evaluating APIs. In their data, lab studies were 16 times more productive than peer reviews. However, the peer reviews were substantially faster and didn't face the same resource problems as lab studies (e.g., recruiting participants and having enough usability experts to conduct the studies).

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 was a user experience containing wild inconsistencies and usability problems. There was no single issue that dominated; rather, users suffered a death from a thousand papercuts.

The web API user experience is affected by all layers between the developer's application and the server that processes their request and deficiencies anywhere in this chain of technical elements can impact the experience, inclusive of the API itself and the server that interprets and executes the API requests. A developer writing code has to work within the constraints and idioms of their programming language. They may or may not use a client library to make using the API easier. This library could be a generic client library (e.g., AngularJS's resource service) or a custom client library written for that API. Another common intermediary is an API proxy. In a browser, the browser itself is yet another intermediary. And of course, the network itself is always an intermediary. Deficiencies anywhere in the stack compel our users to compensate by writing more complex code.

When considering an ecosystem of APIs, such as Google's, the problems are exacerbated by inconsistencies between

APIs. For example, three APIs may expose the concept of a region, but do it in completely different ways (full URL vs. a short name vs. a region code). As a user, even if you have a quick way to map the representation between APIs, do the concepts really mean the same thing in the context of each API? Another example is when you create a new resource. One API may return the newly created resource, another may return a reference to said resource, while a third may return a promise to create the resource. When dealing with an ecosystem, one must consider API usability at a macro level, not just in individual APIs.

Apiness: API Design Reviews at Scale

In 2012, we started the Apiness (API happiness) program in an attempt to address the quality and consistency of Google's APIs. The two initial goals of the Apiness program were to not only identify and fix the most egregious pain points in extant Google APIs, but also to improve the usability of forthcoming Google APIs. In this paper, we focus only on the latter goal; specifically, how it was tackled with an API design review process.

The Apiness review process differs from Farooq et al's aforementioned process. Both processes can be characterized in terms of three key steps (see Table 1).

In our Apiness API design review program, the key stakeholders are:

1. *API Owner.* The stakeholder who is designing and building the API. Although it is usually a team of product managers / engineers, one main point of contact is assigned. All team members do however have full transparency into the process.

	Farooq et al.	Apiness
Kickoff	API owner meets with the usability engineer to set objectives, define reviewer criteria (e.g., prior experience with a particular technology), define scenarios, and produce source code to be reviewed.	API owner requests a review of their API specification and is assigned a design reviewer and a shadow reviewer.
Review	The API owner, usability engineer, and reviewers meet in person and walk through the scenarios (code using the API), providing feedback with the usability engineer moderating to keep the review focussed on usability.	The design reviewer and shadow reviewer review the API specification asynchronously, providing feedback directly to the API owner, and vice versa, iteratively. Issues encountered (for example, insights with broad applicability, or particularly unique / challenging discussion) are brought to a design review team and decisions can be captured in our API style guide.
Exit	The usability engineer and API owner meet to review notes and consolidate into actionable usability defects.	The review goes back and forth until the design reviewer is satisfied, and approves the API. At this point, the design reviewer has the option to graduate the shadow reviewer into a design reviewer for future API reviews.

Table 1: A comparison of the Farooq et al. [7] and Apiness API design review processes.

2. *Design Reviewer.* An experienced API design reviewer who conducts the review and provides guidance.

3. *Design Review team.* The larger team of reviewers, who act as consultants on an as-need basis, providing their collective knowledge and experiences, particularly when the reviewer and the API owner disagree.

4. *Shadow Design Reviewer.* The shadow reviewer is a 'design reviewer in training' whose exact responsibilities are dependent on their progress through the training program. They contribute to the review, without being solely responsible.

5. *Moderator.* The moderator manages the review, ensuring that reviewers are assigned, timelines are adhered to, and the review from the feedback is clear and actionable. They mediate communication, address issues, and drive agenda of the design review team meetings.

The shadow reviewer ensures we continue to scale the program, by providing on the job training in the real world context of reviews. The moderator is an essential addition to ensuring the process scales. As with all large programs, a project manager with a holistic view and understanding is required to keep the process as smooth and efficient as possible.

In addition to extending the roles, our process itself has evolved from that of Farooq et al. The key difference is that the reviews are not conducted as a group-based walkthrough with multiple reviewers, nor are usability insights filed as bugs against the feature owner team. Instead, reviews occur in a more iterative and collaborative fashion. API owners submit the design of their API, sample implementation, code snippets, and relevant supporting material as an online document. Reviewers then leave comments and suggestions directly in the document. This allows for an open dialogue between producer and reviewer, which is particularly critical in situations where specific API domain nuances could lead to misunderstandings or the need for clarifications. This quick and contained back-and-forth directly on the document allows feedback to be contained in a single source, with a trail of all conversation and decision making logic. This approach isn't without fault, as we reflect on in our analysis. Critically, the review remains iterative in nature. Once both API owners and design reviewers agree and sign-off on the API design, reviewers are then automatically assigned as reviewers on any future API changes (e.g. when an API moves from Beta to Generally Available, or if constraints elsewhere force a change to the originally agreed upon design), this provides additional opportunity to review the API, provide guidance on effective design, and ensure that earlier feedback has been considered prior to release, while also maintaining consistency. Critically, this highlights that the review is not done after one linear pass over the document, but rather continues to move and evolve throughout the lifecycle of the API.

In comparison to Farooq et al's. method, which proceeds more like a cognitive walkthrough of an example use of the API (what do you think the next step is? what do you think this code module does? etc), the Apiness review process contains more focus on heuristic evaluation of API design. Samples of end user source code is however essential, as it allows reviewers to see in practice how the API is used, the way in which calls are strung together, the level of abstractions chosen, etc. The Apiness review program uses heuristic evaluation, combined with knowledge of existing Google API designs, to evaluate amongst other things, naming conventions, level of abstraction, error messages, the resource model, the use of standard methods (GET, LIST, PUT, etc), and the use of common design patterns (e.g. pagination, long running operations, etc). Additionally, while also specifically evaluating the API in front of them, as previously mentioned, the Apiness reviewer is looking for consistency with existing Google APIs, and using preexisting knowledge of potential usability issues to drive their reviews. The Apiness team also goes beyond the surface of the API, looking also at the way in which an API will work with Google's programmatically generated client libraries.

The program has now been in place for around 2 years, with a total of 43 APIs receiving review this year. The program, which started with a team of 2 members, has grown to include 18 full reviewers, with 6 shadow reviewers currently in training. Although the program continues to develop and evolve, it has reached a point of maturity. As such, we decided to begin a formal evaluation to evaluate its impact on Google APIs, while reflecting on the execution of the program, and in particular, the challenges involved in implementing API design reviews at scale. In this paper, we discuss early insights gathered from our API producing stakeholders.

Analysis

In order to begin evaluating our design review process, we have initiated a mixed-methods program of investigation which focuses on the experience of our API producing stakeholders, i.e. those individuals / teams that are having their API designs reviewed. This research serves two primary purposes:

1. Better understand how stakeholders perceive the efficacy of the API design process itself.
2. Better understand how stakeholders perceive the impact that the process has on the quality and usability of the API they are building.

Method

In total, since beginning our research, 43 APIs have gone through the API design review process. All of those stakeholders received a survey at the launch of their APIs, which signals the formal end of the design review engagement. Of the 43 stakeholders who went through the review, a total of 39 survey responses have been gathered, for a response rate of approximately 91%.

Each survey, which contains a mixture of quantitative likert-style satisfaction scales and qualitative open-ended questions, gathers data on:

1. Overall satisfaction with the design review [quantitative measure, 5-point likert].
2. Reasoning behind overall satisfaction level [qualitative].
3. Perception of impact on API quality [quantitative measure, 5-point likert].
4. The value (or lack thereof) of the design review process [qualitative].
5. Additional comments or suggestions on any aspect of the design review process [qualitative].

Additionally, informal semi-structured interviews have been conducted with a small subset of API producing stakeholders. These interviews cover the same broad themes, but aim to provide richer input into the experience. In total, data from 2 stakeholder interviews are considered in this paper.

These interviewees, who also completed the survey, were selected via a convenience sampling approach. Their survey data had not been viewed or analyzed prior to interview.

Results
Overall, stakeholders who had their API reviewed were generally satisfied with the design review process. 29 respondents (approximately 75%) rated themselves as either completely or somewhat satisfied (the top 2 buckets of the likert-style scale). Only 3 respondents rated themselves as dissatisfied.

With respect to perceived impact on quality, 35 of the respondents (approximately 91%) felt that the overall quality of their API had improved post review, with about one third (14 respondents) stating that their API was much better (the top bucket on the likert-style scale). Only one stakeholder felt that their API post review was, overall, lower quality.

We thematically coded open-ended qualitative feedback to better understand what lead API design producers to score the design review process, and its implications on the quality of their API, as they did. First, we focus specifically on the positives, and in particular on three key themes that emerged from the data:

Improved API Surface
Consistent with the quantitative data, and key to the overall goal of the program, many of the API producers reflected that their API design had improved.

"[Reviewer] was great in finding issues with the API"

"The design review gave us a lot of actionable feedback"

Greater consistency across Google's API Ecosystem
Again, key to the overall goal of the program, API producers consistently reflected that their API was now more consistent with other Google APIs, to the greater benefit of Google's API ecosystem.

"The [sign-off] from the reviewer gave me the comfort in knowing that our API would have the same consistent look-and-feel as the other Google APIs."

"The most useful part is that it produces uniformity across the whole set of Google APIs."

Validation from Experts
Even in cases where the API producer felt that their API design had not changed considerably, validation from an external API design expert was welcomed, giving the API owner confidence in their design.

"The design review was a nice sanity check that our APIs are understandable to people who didn't write them."

It is positive that API producers saw value in the design review program, and consistently identified the primary goals of the program in their positive reflections. These initial insights provide early validation that the design review process can be effectively utilized at scale, to help API producers build usable APIs, that maintain consistency despite a heavily federated build process.

Reflections on Scaling API Design Reviews

Despite these positives, it is important to reflect on areas for improvement as we continue to grow and iterate on the program. Thematic coding of all open-ended responses identified a number of themes for reflection. One obvious direction for this analysis would be to focus on those API producers who indicated in the scale questions that they were dissatisfied with the process, or its impact on their API. However, due to the small number of respondents who fit into this category, it is difficult to tease out specifics.

Anecdotally, we notice that the feedback given by dissatisfied respondents fits consistently into the broad themes outlined below. In this paper we highlight four particular issues, selected due to their prevalent occurrence in our feedback.

Keeping Feedback Consistent
Key to any review process is consistency in advice and guidance. Inconsistent messages cause frustrating time delays and impact the trust one has in the quality of the review process. When scaling the API design review process to multiple APIs and reviewers simultaneously, it is important to ensure the advice and guidance given is coherent and consistent. This is particularly true given the dependencies that exist between APIs, and the fact that API producing stakeholders are likely to go through the process multiple times, particularly for products that expose multiple APIs to their customers. While satisfaction was broadly high, lack of consistency was regularly mentioned in our qualitative feedback as negatively impacting the experience.

"there was a particularly frustrating cycle of reviews around naming conventions where I received contradicting messages"

"As an example, during the course of our API being developed the best practice on pagination tokens went from string to bytes and finally back to string"

"the feedback coming from various folks on the [design review] team often conflicted with each other"

Currently, the design review team has bi-weekly meetings to discuss the design review process and API design best practice. Additionally, when in doubt, design reviewers can use the wisdom of the crowd, and take issues to the entire review team. Additionally, new reviewers go through a shadow review process before individually reviewing APIs. Despite all this, the team faces the challenge of ensuring high quality review, while continuing to meet demand in a timely fashion.

To help scale, the design review team has undertaken two initiatives. The first is increased documentation, and thus transparency, into the design review process. When one reviewer identifies a usability issue, and provides guidance on improved design, this should be centrally documented so that other reviewers are clear on the advice their peers are giving, and API producing stakeholders, who may still be designing their API, can ensure they themselves do not fall for the same usability pitfalls. Documenting in this way keeps the API reviewers honest, and ensures API producers can get design advice before submitting their review. The second initiative is API design classes. By taking the wisdom of good API design and running regular classes, we can help API producing engineers avoid common design mistakes / challenges before they begin their API design. Synthesizing advice into materials for a class gives reviewers the time to reflect on best practice, while also providing API producers with one source of truth for advice prior to designing their API. Raising the bar of the APIs proposals prior to design review submission makes the review process itself simpler and more timely, the latter of which is another key challenge for scaling API design reviews.

Time is Money
Time very literally is money when it comes to launching a new product. As such, it is critical that an API design review process is not seen as a blocker. This is really comprised of two factors. First, the value of the API design review process must be established, such that API producers are clear that it is necessary, and not simply a zero-sum game. Rather, any delay to launch will be outweighed by the user/customer benefit derived from the improved qual-

ity of the API. With this in mind though, API design reviews should still be as timely and efficient as possible. The single most frequent theme in our data was the time required for the API design review.

“The design review process took very long for us. The review was helpful, but just hope it was faster.”

“It took a while to get the API review started, and this caused some delays on our end as people who had bandwidth to help with implementation became busy and unable to help.”

As previously discussed, the team is already working on initiatives to help raise API design quality and set expectations for the process, through clear documentation and API design classes. However, beyond this, given the scale of Google’s API ecosystem, it is worth considering whether other measures can be implemented. For example, if a new version of an API is being released, where 90% of the surface remains unchanged, how should that review look? Is this an opportunity to revisit the API, and ensure current best practice is in place, or can a lightweight review be constructed? If an API is a single resource and method, should there be a different approach to that for a 100 method API? Likewise, if an API will likely never see more than 10 external users, how should the review look compared to an API which is targeting hundreds of thousands? If a context dependent review process is initiated, how can we ensure consistency (in both process and API design) while clearly setting expectations for API producing stakeholders? Future work will aim to measure the impact of the two initiatives currently in place, and assess the utility of the other approaches discussed in this section.

API Expertise Vs. Domain Expertise

All of Google’s major product areas, from Cloud Platform to Maps, offer APIs, each of which serves its own audience and use cases. With over 100 public APIs already available, and an ever growing list in the pipeline, it is increasingly difficult for API design reviewers to be knowledgeable on all services. While the reviewer is recruited and trained on API design best practice, providing design advice inherently requires a baseline level of understanding of the service. We see in our data the emergence of a tension between expertise on API design and domain level expertise on the product area of the API.

“The beginning of the process was very frustrating. We had a hard time getting the review team to understand what we were trying to accomplish in our API”

“It would be nice if there would be reviewers from major product area such as [X] as most of the time was spend in describing how [product X] works.”

As Beaton et al. discuss, applying traditional usability heuristics to API evaluation requires careful consideration [1]. It is important to question the extent to which it is possible to construct a fully centralized API design review process, where reviewers are tasked with reviewing and providing guidance on a drastically diverse array of APIs. Is it possible to provide a centralized set of design guidance that is abstract enough to be suitable to diverse APIs, while concrete enough to ensure consistent interpretation and application? Certainly, our early survey results indicate API stakeholders are, for the most part, satisfied with the process. However, qualitative responses indicate that this is an issue that we must keep an eye on.

“The only downside that we experienced from this was the necessity to conform to specific REST styles. In some ways, this limited our API. Since our API doesn’t fit the cookie cutter mold of a RESTful API we had to make some design concessions.”

"...part was that the use-case for our API differed so strongly from what I think the process was intended for."

Dependent Services
One interesting challenge when scaling a design review program up to multiple simultaneous API reviews is that there is potential for dependencies between APIs currently under-review. As such, recommendation for change in one place can have a cascading effect on other services, services which themselves may be in-flux through the design review.

"There were several changes, though, which wound up creating enormous amounts of work in the [x] client and service."

This adds complication to the design review process, and requires careful coordination between API stakeholders and design reviewers. In an ideal world, the bottom level APIs would be reviewed and approved before any other services were built on top of them, however in practice, this isn't always feasible due to time or resource constraints. Given this, more emphasis is placed on getting things 'as right as possible' the first time. The aforementioned API design classes, and documentation of previous reviews and outcomes should help API producers ensure their API does not fall into any big traps before initiating the review process, or building dependent services on top of them.

Next Steps and Future Work

In this paper we have provided early insights into our Apiness API design reviews at scale program. Early indication from our API producing stakeholders suggest the program has been successfully implemented, with consistent positive sentiment towards the goals of the program and the impact it has on Google's APIs. One open question remains though, are these APIs really more usable to the external developers? Apiness also has the goal of better understanding what makes an API more usable. Work is currently underway to assess the usability of the APIs that have gone through the design review process, and benchmark performance against those APIs that were released pre API design review program. A mixture of API log file analysis, high-touch usability studies, and survey data are being employed, although the work is too early in nature to report on here.

Additionally, outlined in our results, we highlight four key areas for further reflection. It is important to also stress, our API program continues to grow, with ever more APIs being built for both internal and external consumption. As such, we face a number of key challenges, which are inherently related to the themes identified in this paper. In particular, 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? Further, with more APIs, comes greater diversity in product, service, use-case, and target audience. With a centralized review process, can we continue to ensure Google wide API consistency? And, does a one-size-fits-all review process make sense? If a usable API in one domain looks very different to that in another, would a more federated process, where reviewers are both domain specific experts and API design experts, lead to more usable APIs? Additionally, with a growing field of work assessing programatic evaluation of API usability, we continue to explore whether the wisdom of the API design team can be captured in our tooling. One exploration includes the use of linting as a first pass over an API design, in order to identify common / low-hanging API design issues before human assessment is required.

In our future work, we will continue to analyze our survey responses, while iterating on our design review program to ensure API usability and API consistency are top of mind for Google APIs. Additionally, future work will aim to measure the resultant impact on the end-developer who works with the APIs, to assess whether our API design review program is really having the influence both reviewers and API producing stakeholders believe.

Conclusions

In this paper we introduced Apiness, a program designed to improve the usability of Google's APIs. We reported on one particular facet, API design reviews at scale. Building upon prior work in the field, we implemented a process to review APIs for usability and consistency issues, and did so in a way that scales to the number and diversity of Google's APIs. Early indications suggest the program has started successfully, although further work remains to fully validate the usability impact on the APIs being produced. This paper contributes to the field an overview of our implementation of the program, as well as the key lessons learned and challenges others may face when implementing an API design review program of their own.

References

[1] J. Beaton, B. Myers, J. Stylos, S. Y. Jeong, and Y. Xie. 2008. Usability Evaluation for Enterprise SOA APIs. In *Proc. of the 2nd International Workshop on Systems Development in SOA Environments (SDSOA '08)*. ACM, 29–34.

[2] K. Beck and C. Andres. 2004. *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional.

[3] J. 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 (OOPSLA '06)*. ACM, 506–507.

[4] A. Bosu, M. Greiler, and C. Bird. 2015. Characteristics of Useful Code Reviews: An Empirical Study at Microsoft. In *Proc. of the International Conference on Mining Software Repositories*. IEEE.

[5] S. Clarke. 2004. Measuring API usability. Dr Dobb's Journal. (2004). http://www.drdobbs.com/windows/measuring-api-usability/184405654

[6] M. E. Fagan. 1999. Design and Code Inspections to Reduce Errors in Program Development. *IBM Syst. J.* 38, 2-3 (June 1999), 258–287.

[7] U. Farooq, L. Welicki, and D. Zirkler. 2010. API Usability Peer Reviews: A Method for Evaluating the Usability of Application Programming Interfaces. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, 2327–2336.

[8] T. Grill, O. Polacek, and M. Tscheligi. 2012. Methods towards API Usability: A Structural Analysis of Usability Problem Categories. In *Human-Centered Software Engineering*, M. Winckler, P. Forbrig, and R. Bernhaupt (Eds.). Lecture Notes in Computer Science, Vol. 7623. Springer Berlin Heidelberg, 164–180.

[9] G. M. Rama and A. Kak. 2015. Some structural measures of API usability. *Software: Practice and Experience* 45, 1 (2015), 75–110.

[10] J. Stylos and S. Clarke. 2007. Usability Implications of Requiring Parameters in Objects' Constructors. In *Proc. of the 29th International Conference on Software Engineering (ICSE '07)*. IEEE, 529–539.

[11] J. Stylos and B. Myers. 2007. Mapping the Space of API Design Decisions. In *Proc. of the IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '07)*. IEEE, 50–60.

[12] L. Williams and R. Kessler. 2000. All I Really Need to Know About Pair Programming I Learned in Kindergarten. *Commun. ACM* 43, 5 (May 2000), 108–114.