API Peer Reviews: A Method for Evaluating Usability of Application Programming Interfaces

Umer Farooq and Dieter Zirkler
Microsoft Corporation
1 Microsoft Way, Redmond, WA 98052 USA
{umfarooq, dieterz}@microsoft.com

ABSTRACT
API usability tests in the lab are time and resource intensive, thus allowing a relatively small percentage of the API namespace to be evaluated. We describe a group-based usability inspection method – API Peer Reviews – to evaluate API usability. Based on an analysis of usability breakdowns from API Peer Reviews and API usability tests, results show that API Peer Reviews identified breakdowns across several cognitive dimensions, some of which were different than what was identified by API usability tests. We reflect on the adoption of API Peer Reviews as a collaborative practice in organizations for evaluating API usability.

Author Keywords
API usability, cognitive dimensions, usability inspection.

ACM Classification Keywords
H.5.3 Group and Organization Interfaces: Evaluation/methodology.

General Term
Human Factors

INTRODUCTION

Designing usable APIs is critical for any software organization exposing a programmatic user interface. The most compelling reason is that highly usable APIs can drive adoption and sustained use of a particular technology. Initial encounter with a poorly designed API can leave a lasting impression of complexity and discourage users from ever adopting a particular technology [4].

Though many heuristics and guidelines exist for designing usable APIs [2, 4], the variety of methods available for evaluating API usability is limited. The most common and well-documented method in literature for evaluating API usability is empirical testing, that is, formal API usability tests in the lab (e.g., [9]). However, this method is not scalable. Real users can be difficult or expensive to recruit in order to test all aspects of a particular design artifact [7, p. 2]. When the design artifact is an entire API (e.g., Microsoft's .NET framework) comprising hundreds of namespaces and libraries, usability tests cannot practically provide adequate API test coverage. With one or two usability engineers supporting API usability tests for the entire framework, only a small percentage of the APIs can practically undergo usability evaluation.

Usability inspection methods, such as programming walkthroughs [1], have been proposed to address the shortcomings of formal empirical evaluations. For user interface design, usability inspection methods have been shown to identify many usability problems that are overlooked by formal empirical evaluations, which suggest that the best results are achieved by combining empirical tests and inspections.

In this paper, we describe an inspection method for evaluating the usability of APIs. API Peer Reviews provide an efficient and scalable way for product teams to get usability feedback on APIs. We present preliminary results of comparing usability breakdowns between API Peer Reviews and API usability tests conducted at Microsoft.

METHOD DESCRIPTION

The goal of an API Peer Review is to get usability feedback on a particular API feature. In the .NET framework, for example, the System.Net.Sockets namespace would be considered an API feature that provides developers the ability to tightly control access to the network.

Roles

An API Peer Review is a group-based usability inspection method comprising four interdisciplinary roles.

Feature owner is the person who owns the particular API feature to be reviewed. This is typically the program manager who writes the feature specifications. The role of the feature owner is to lead the API Peer Review.

Feature area manager is the person who owns the overall feature in general. This is typically the feature owner's manager who has breadth knowledge of how the particular API feature to be reviewed relates to other API features he/she is managing. The role of the feature area manager is to record feedback as notes during the API Peer Review.

CSCW 2010, February 6–10, 2010, Savannah, Georgia, USA.

Usability engineer is the person who is responsible for evaluating API usability for the feature area. The role of the usability engineer is to facilitate and moderate the API Peer Review discussion from an API usability perspective.

Reviewers are organizational peers (hence the name API Peer Reviews) of the feature owner who provide usability feedback on the API feature to be reviewed. In each API Peer Review, the goal is to have 3-4 reviewers. These reviewers are other feature owners, software engineers, and testers not fully familiar with the API feature but are considered knowledgeable enough to provide feedback.

Process (in three steps)

(1) Planning: The usability engineer and feature owner kick off the process in a 1-hour meeting to plan the API Peer Review. The goals of this meeting are the following:

- Define the objectives. These are 1-2 key questions that the feature owner wants feedback on from the API Peer Review. For example, in the case of System.Net.Sockets, the key question may be "How easy is it to pass a message to a server using the Winsock interface".
- Identify code sample. The API Peer Review is driven by walking through a code sample. The feature owner either identifies an already existing code sample or harvests code snippets from the feature specification.
- Choose reviewers. The feature owner specifies the criteria for reviewers. For instance, one criterion could be that reviewers should have two years of C# experience.
- Decide logistics. The usability engineer and feature owner work together to recruit reviewers and specify time and location for the API Peer Review.

(2) Review: The duration of an API Peer Review is 1.5 hours in a meeting room with everyone (all the roles) synchronously present face-to-face. The usability engineer takes the first 5 minutes to enumerate the goals of the API Peer Review to the reviewers. The feature owner takes the following 10-15 minutes to explain the background of the API feature under review so all reviewers have the same baseline context in order to provide usability feedback. The remaining time is dedicated to the actual review itself.

The method of API Peer Reviews is an adapted version of cognitive walkthroughs [10]. API Peer Reviews tend to focus on ease of learning an API and ease of discoverability of API classes, methods, and parameters. In API Peer Reviews, the feature owner walks the reviewers through a series of actions that a target developer would complete to perform tasks with the API under review. For each step, reviewers are prompted to consider the mismatches between the API designers' conceptualization and their own, poor choices of names (e.g. a method name), inadequate exposure of functions by the API to discover its underlying mechanics, and so forth. The usability engineer anchors, leverages, and converges the feedback by the reviewers toward API usability problems.

The review process is a group-based walkthrough of the code sample. For example, the System.Net.Sockets code sample may have two methods: receiveMessage and sendMessage. The review is driven by two questions: (1) What do you think this code snippet (e.g., for a method) does, and (2) Does the implementation in the code module make sense? The first question probes the reviewers in thinking about what a code snippet is doing. This typically leads to a discussion with the feature owner if the code snippet is difficult to comprehend. Once the reviewers understand what the code snippet is doing, the second question solicits specific feedback on implementation details. The usability engineer facilitates this discussion in order to solicit detailed usability feedback. For example, one of the reviewers may comment "The name of the method doesn't make sense". In this case, the usability engineer asks standard think-aloud probing questions ("why do you think the name doesn't make sense") and tries to converge the discussion toward actionable feedback ("can you suggest method names that would make more sense").

(3) Bug filing: The usability engineer, feature owner, and feature area manager meet for 1 hour after the API Peer Review (typically the next day) to record the usability breakdowns as bugs. The notes recorded by the feature area manager are used as grist for specifying the usability bugs. The role of the feature owner in this meeting is to open/enter the bugs. The role of the feature area manager in this meeting is to specify any additional context, feasibility, and priority of the bugs. The role of the usability engineer is to ensure that all actionable usability feedback is captured.

CONDUCTING API PEER REVIEWS IN THE FIELD

Our product team has been actively involved in conducting both API usability tests (see [3] for process) and API Peer Reviews for the .NET framework at Microsoft. The outcomes of both these evaluations are usability breakdowns that are entered as bugs in a central bug repository. We wanted to understand the nature of bugs that came out of API Peer Reviews and API usability tests.

Using the cognitive dimensions framework

We adopted the cognitive dimensions framework [5] to analyze bugs. The cognitive dimensions framework, rooted in the psychology of programming, has been successful for evaluating API usability.

The cognitive dimensions framework presents a set of 12 different dimensions that have a usability impact on the way that developers work with an API and on the way that developers expect the API to work, which makes is highly suitable for analyzing API usability feedback. We use these dimensions to categorize the usability bugs from API Peer Reviews and API usability tests. (Readers not familiar with the 12 cognitive dimensions should read the article in [3].)

Usability bugs

We analyzed 57 bugs: 29 bugs from API Peer Reviews and 28 bugs from API usability tests. These bugs represent usability breakdowns on part of the .NET API found through 5 API Peer Reviews and 2 API usability tests over a 6-month period. This data is an opportunistic sample and thus does not reflect any explicit sampling procedure. Our minimum requirement was to analyze bugs from at least two API Peer Reviews and two usability tests.

Data analysis

Two judges, each familiar with the cognitive dimensions framework, categorized each bug as part of two iterations. During the first iteration, the two judges categorized each bug independently. During the second iteration, the two judges discussed disagreements from the first iteration, deliberated over the disagreements, and tried to come to a consensus. If consensus was not achieved, a third judge independently resolved each disagreement.

RESULTS AND DISCUSSION

At the end of the first iteration, there was a 70% agreement rate. After the second iteration, a 95% agreement rate was reached (consensus was not achieved on 3 bugs). The third judge comfortably resolved the remaining disagreements. Figure 1 illustrates the results of bug categorization. As the number of bugs was unequal from the API Peer Reviews (29) and API usability tests (28), they were normalized with respect to the total number of bugs for each method. Thus, percentages are reported.

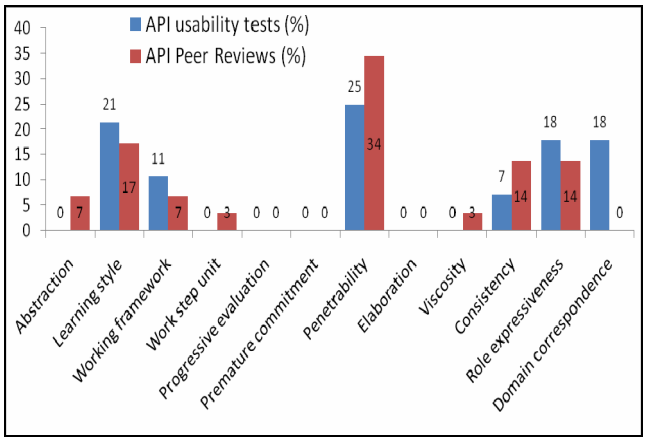

Figure 1. Mapping of bugs onto cognitive dimensions.

Nature of bugs from API Peer Reviews

API Peer Reviews identified a range of bugs across many of the cognitive dimensions. In particular, API Peer Reviews were highly sensitive (greater than 3 bugs) to identifying bugs related to learning style, penetrability, consistency, and role expressiveness. Our result implies that API Peer Reviews are indeed useful, as the bugs did in fact relate to the various API usability problems manifested as cognitive dimensions in the framework.

Comparison of bugs

Even though the API Peer Reviews and API usability tests did not evaluate exactly the same API features, both methods did evaluate API features in the same area (Windows Workflow). With this caveat in mind, we wanted to understand whether or not API Peer Reviews identified different usability problems than API usability tests.

API Peer Reviews identified bugs not identified by API usability tests, related to abstraction, work step unit, and viscosity. Usability bugs related to abstraction reveal the difference in the number of components a developer expects to use in order to accomplish a task versus the number of components the API exposes. Usability bugs related to work step unit reveal a developer's expectation of how much of a programming task can be completed in a single step. Usability bugs related to viscosity reveal the expectation of a developer in how much effort he/she needs to expend to make a code change using the API. These three categories of usability bugs explicitly refer to ease of learning the API during first exposure, focusing on developers' expectations of correct actions based on perceived similarity to the developers' current goals. Hence, such categories of usability bugs are more likely to be apparent as first-order feedback when developers look at and review an API, where they do not have an opportunity to overcome learning barriers by actually using the API during API usability tests. This result is consistent with the underlying theories of skill acquisition behind cognitive walkthroughs [10]. On the contrary, API usability tests identified bugs not identified by API Peer Reviews, related to domain correspondence. Usability bugs related to domain correspondence reveal how clearly API components map to any special domain-specific tricks that a developer needs to be aware of in order to accomplish a task. Such domain-specific tricks are likely to be revealed when a developer is actually writing code as opposed to reviewing code.

An interesting result was that both API Peer Reviews and API usability tests did not identify certain categories of bugs, related to progressive evaluation, premature commitment, and elaboration. It is possible that such bugs did not exist, or if they did exist, they were not identified as bugs. The latter explanation seems more plausible as neither of the methods is tuned to identify bugs related to the missing categories. Usability bugs related to progressive evaluation reveal to what extent partially completed code can be executed to obtain feedback on code behavior. Such behavior is typically not captured as a usability bug but rather as a passing observation of the developer testing code before it is complete. Usability bugs related to premature commitment reveal the amount of decisions that developers have to make when writing code, which is quite challenging to probe and elicit. Usability bugs related to elaboration reveal the extent to which an API feature must be customized and adapted to accomplish a task. In both API Peer Reviews and API usability tests, the developers are typically expected to use the API features as is without

customizing or extending its functionality. Both methods may need to be refined in order to increase the sensitivity of usability bugs not identified by either method.

In summary, API Peer Reviews are sensitive to revealing usability bugs related to cognitive dimensions that explicitly reveal a developer's expectations about ease of learning in using an API. Echoing Greenberg and Buxton's [6] argument, this suggests that API Peer Reviews and API usability tests can be used in conjunction to evaluate API usability at different points in a design cycle, depending on the particular usability goals.

Limitations

We used a non-empirical analytical method (cognitive dimensions) to compare the usability breakdowns from both methods. As future work, quantitative analysis can more definitively characterize the effectiveness of API Peer Reviews. For a more direct comparison, we will also conduct both methods on the same API feature.

ORGANIZATIONAL ADOPTION AND SCALING

Our paper describes a method to evaluate APIs, which is an under-examined key technology supporting the collaborative development of software that facilitates work and other activities (Web 2.0 is a great example). Of particular interest to the CSCW community is how API Peer Reviews can be adopted and scaled in organizations.

The collaborative practice of socially reviewing APIs aligns well with existing software design culture. Our motivation to conduct API Peer Reviews in a group setting was driven by the advantages of group-based cognitive walkthroughs (see discussion in [7, pp. 9-10]). However, the real value of collaboratively reviewing APIs in a group setting was that software development teams could relate to API Peer Reviews based on their prior experience with collaborative software engineering practices such as agile software development aka SCRUM [8]. Within Microsoft, a number of product teams emphasize frequent, deep, and iterative communication between the multiple disciplines involved in designing, implementing, testing, and releasing products. Streamlining SCRUM-based practices to review APIs seemed to be a smooth transition for software development teams at Microsoft.

The most telling characteristic of API Peer Reviews is their minimal time investment. As a consequence, API Peer Reviews provide broader usability test coverage, making them relatively scalable to API usability tests in the lab. The minimal time and resources (no cost to recruit users; no lab equipment needed) required to conduct API Peer Reviews have made the method popular with the product teams we are working with at Microsoft. As evidence from our first-hand experience, we can state that the subset of the .NET team that we work with has adopted API Peer Reviews and has incorporated the method as part of the core release exit criteria for usability on their APIs.

In addition to the high benefit-to-cost ratio for API Peer Reviews, we argue that sustainable organizational adoption and scaling of the method will take place by developing and maturing best practices around API usability. Our method suggests a collaboration style in which product teams are implicitly being coached to become more sensitized in identifying API usability breakdowns. This is because the product teams are gaining valuable API usability design knowledge as a social outcome of participating in the process (reviewers) and leading the session (feature owner). The goal is to harvest and abstract such knowledge over time in the form of design guidelines that eventually become an integral part of the organization's design practices around API usability.

ACKNOWLEDGEMENTS

We thank Jonathan Grudin and Brian Bailey for guiding the paper revision. A special thanks to Bob Schmidt and Leon Welicki for helping develop the method. We are grateful to Guoping Ma and Steven Clarke for their analytical insights.

REFERENCES

1. Bell, B. *Using programming walkthroughs to design a visual language*. Ph.D. dissertation, University of Colorado, 1992.
2. Bloch, J. How to write a good API and why it matters. *Keynote address for LCSD workshop at OOPSLA*, 2005. http://lcsd05.cs.tamu.edu/#keynote.
3. Clarke, S. Measuring API usability. *Dr. Dobbs Journal* (May 2004), http://www.ddj.com/windows/184405654.
4. Cwalina, K. and Abrams, B. *Framework design guidelines*. Addison-Wesley, 2005.
5. Green, T.R.G. and Petre, M. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *Journal of Visual Languages & Computing 7*, 2 (1996), 131-174.
6. Greenberg, S. and Buxton, B. Usability evaluation considered harmful (some of the time). *Proc. CHI 2008*, ACM Press (2008), 111-120.
7. Nielsen, J. and Mack, R.L. *Usability inspection methods* (Ed). John Wiley & Sons, 1994.
8. Schwaber, K. and Beedle, M. *Agile software development with SCRUM*. Prentice Hall, 2002.
9. Stylos, J. and Clarke, S. Usability implications for requiring parameters in objects' constructors. *Proc. ICSE 2007*, ACM Press (2007), 529-539.
10. Wharton, C., Rieman,J., Lewis, C. and Polson, P. The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R.L. Mack, *Usability inspection methods*, John Wiley & Sons, 1994 (pp. 105-140).