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

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

ABSTRACT
We describe a usability inspection method to evaluate Application Programming Interfaces (APIs). We found the method useful as it identified usability defects in part of Microsoft's .NET Framework, of which 59% were new and 21% were fixed. Based on a comparison of usability defects identified between API usability peer reviews and API usability tests, API usability tests were found to expose design issues related to actually using an API whereas API usability peer reviews were found to expose the design rationale of an API. We reflect on the efficiency and productivity of each method: each API usability test is equivalent to approximately 16 API usability peer reviews. We discuss how API usability peer reviews can be used in conjunction with API usability tests to increase usability coverage on APIs.

Author Keywords
API usability, usability inspection, usability evaluation method (UEM), software bugs, usability breakdowns.

ACM Classification Keywords
H.5.2 User Interfaces: Evaluation/methodology.

General Terms
Design, Human Factors.

INTRODUCTION
Designing usable APIs is critical. The most compelling reason is that usable APIs can drive adoption and sustained use of a particular technology [4, 19]. One of the challenges that software organizations face is that they do not have an army of usability engineers to provide usability coverage for all the APIs the organization is producing. Typically, a handful of usability engineers are supporting the usability evaluation of large APIs comprising thousands of classes.

In this paper, we describe an inspection method to evaluate the usability of APIs. *API usability peer reviews* provide a low cost, efficient, and scalable way for product teams to get usability input on APIs. Based on quantitative data and analysis, our research shows that the method is useful in identifying new API usability defects. Further, API usability peer reviews identify usability defects that are different from the ones identified through standard API usability tests in the lab, thus implying that the two methods complement each other.

This paper is organized as follows. We first review related work on API evaluation methods. We then describe the method of conducting API usability peer reviews. We proceed to the design of our research study, show the usefulness and comparative feasibility of API usability peer reviews, and outline implications of using the method.

CHI 2010, April 10–15, 2010, Atlanta, Georgia, USA.

RELATED WORK

Software engineering
The notion of reviewing APIs is not new in software engineering. In the mid-1970s, Fagan pioneered inspections to identify defects in software [10]. Related inspection methods, such as software peer or code reviews [1, 10, 11, 13, 24, 32, 35, 40, 42], follow Fagan's method of using inspectors to review technical content and quality, such as performance issues, memory leaks, and architectural tradeoffs, but not necessarily usability. Our proposed method focuses exclusively on the usability of an API. During software reviews, the code that makes up and exposes the API is typically inspected. On the other hand, API usability peer reviews inspect the public surface of the API that end user developers will interact with.

Human Computer Interaction
In HCI literature, the most common and well-documented method for evaluating API usability is empirical testing, that is, API usability tests in the lab (see [36] for an example of an API usability test). While usability tests in the lab provide deep feedback based on real users interacting with the interface, they have many drawbacks [9, 21, 27, 33]. For API usability tests specifically, software developers with specialized domain knowledge can be difficult to recruit. Another drawback is that usability tests take considerable time — to plan the study, conduct the user sessions, and analyze data — in the order of several weeks. This can present a significant challenge to product teams who, depending on the stage of the software development lifecycle, often require quick feedback on the usability of APIs in a matter of days rather than weeks or months. Further, usability testing is expensive (e.g., cost of recruiting users and using lab facilities).

Usability inspection methods address the drawbacks that are characteristic of usability tests in general [28]. Two common usability inspection methods are heuristic evaluations and cognitive walkthroughs. API usability peer reviews are distinct from heuristic evaluations, which typically require usability specialists to judge the compliance of the artifact with recognized usability heuristics [27]. For APIs (in contrast to GUIs), such usability specialists are not commonly available because of the niche domain, in which case extensive training is required. By adding an API usability engineer as a facilitator in our proposed method, we get some benefits of heuristic evaluations without the need for lots of training. API usability peer reviews are also different from cognitive walkthroughs. While cognitive walkthroughs focus on ease of learning [41], API usability peer reviews also focus on other aspects of usability specific to APIs, such as consistency and role expressiveness [16]. API usability peer reviews are also distinct from programming walkthroughs. Programming walkthroughs [2, 3] use usability specialists and API designers to review the underlying code, but API usability peer reviews use inspectors or reviewers outside of the immediate team building the API.

Most notably, what distinguishes API usability peer reviews from traditional usability inspection methods is their feasibility to scale the evaluation of APIs in large software organizations. For example, in the product team that we work with at Microsoft, there is one usability engineer supporting a product team of over one hundred developers, testers, and program managers designing a large number of APIs. First of all, traditional usability inspection methods relying on multiple usability specialists to evaluate APIs do not apply to such an organizational context, as we have one usability engineer available. Second, traditional usability inspection methods advocate a consultant-based approach where usability engineers provide services to a product team, which yields the usability engineer as a bottleneck for knowing and applying usability best practices.

Our proposed method suggests an apprenticeship-based approach in which usability engineers actively engage the product team in a social process of reviewing APIs. By adopting a Vygotskian approach [39], our method implicitly trains the product team to (a) become sensitized to usability best practices, and (b) incorporate such tacit knowledge (usability best practices) as an integral part of their design practice. The outcome of API usability peer reviews is the same as traditional usability inspection methods (i.e., the identification of usability defects). However, our method also coaches the product team on API usability best practices that can eventually be institutionalized as design guidelines (e.g., [6]). API usability peer reviews reflect the adage — "Tell me and I forget, show me and I remember, involve me and I understand" — to scale the method at an organizational level.

Comparison of usability evaluation methods

Of particular interest to one of the contributions in this paper is the literature on comparison of usability evaluation methods (UEMs). For example, Jeffries and colleagues [22] compared heuristic evaluation, software guidelines, cognitive walkthroughs, and usability testing. They found that heuristic evaluation identified the most serious usability defects — which were not identified through guidelines or walkthroughs — with the least amount of effort. However, usability testing identified more recurring and high-priority defects. Several other papers, similar to [22], compared UEMs [5, 9, 18, 23, 25, 29, 33]. One of the generalized results from these studies is that usability inspection methods identify many usability defects that are overlooked by usability tests in the lab and vice versa, which suggest that the best results are achieved by combining empirical tests and inspections [21, 27]. Our empirical contribution in this paper builds on this research.

While it is not our goal to review the literature on comparison of UEMs, it is worthwhile mentioning the debate sparked by Gray and Salzman [15]. They reviewed a series of studies that compared UEMs, criticizing them on a number of dimensions such as interpretations of causality, issues of generality, and validity of conclusions. This led to a commentary by Olson and Moran [30] in which several leading researchers and practitioners contributed their thoughts on the debate. Our intention in this paper is not to touch this particular debate. Instead, we describe a usability inspection method to evaluate APIs and empirically contribute to the understanding of the usefulness and comparative feasibility of our proposed method.

DESCRIPTION OF API USABILITY PEER REVIEWS

In this section, we explain the roles involved in and overall process of conducting an API usability peer review. We provide a field example so readers gain practical insights into how to apply this method in their own organizations.

Roles

(1) Feature owner is the person who owns a particular feature of an API that is to be evaluated from a usability perspective. This is typically a program manager who writes the specifications for a particular feature of the API. The primary role of the feature owner is to determine the goals and content of the API usability peer review, drive the review session, and identify the usability defects.

(2) Feature area manager is the person who owns the overall feature area of the API. This is typically the feature owner's manager who has broad knowledge of how the particular API feature under review relates to other API features he/she is managing. The primary role of the feature area manager is to fill any knowledge gaps that the feature owner is not aware of and to record feedback as raw notes during the API usability peer review session.

(3) Usability engineer is the person who is responsible for evaluating the usability of the API feature under review.

The primary role of the usability engineer is to assist the feature owner in defining the goals and content of the API usability peer review, coordinate the recruitment of reviewers, facilitate the review session, record feedback during the API usability peer review session, and ensure that the feedback is captured as usability defects.

(4) Reviewers are organizational peers (hence the name API usability "peer reviews") of the feature owner who are not fully familiar with the API feature under review but are considered knowledgeable enough to provide feedback. These are typically program managers, software developers, and software testers working on features other than the one under review. Reviewers are recruited through distribution lists and word of mouth. They are identified based on how close their skills and experiences resemble those of targeted end users. Three to five reviewers are recruited, as we have found this range to be ideal for a productive API usability peer review. The primary role of the reviewers is to provide usability feedback on the API feature.

Process overview

The process of conducting an API usability peer review consists of three phases (Figure 1).

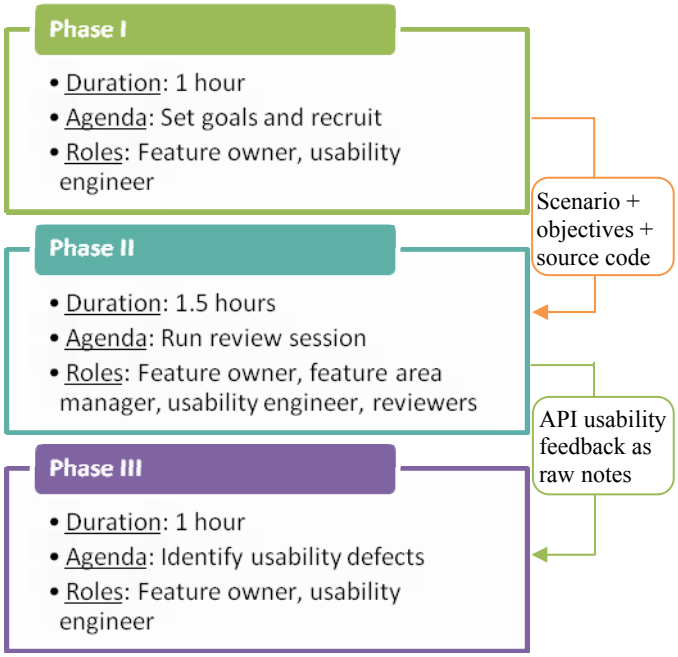

Figure 1. Overview of API usability peer review process.

Phase I: Goal setting and planning. The usability engineer and feature owner initiate the process in a 1 hour meeting. The purpose of this meeting is the following.

- *Pick scenario*. The first agenda item of the meeting is to identify the primary usage scenario for the API feature that reviewers will walk through during the API usability peer review session. The scenario typically consists of core tasks that a user is expected to complete using the API feature.
- *Define objectives*. These are key questions that the feature owner wants feedback on from reviewers.
- *Identify source code*. An API usability peer review is driven by walking through source code that illustrates the scenario. The feature owner identifies an already existing code sample, or harvests code snippets or even early pseudo-code from the feature specification.
- *Specify reviewer criteria*. The feature owner specifies the criteria for reviewers, such as prior experience with a particular technology. The incentive for reviewers is the expansion of their knowledge in an API feature they are interested in and an opportunity to interact with the product team (feature owner and feature area manager).
- *Decide logistics*. The usability engineer and feature owner work together to recruit reviewers and specify time and location (e.g., conference room) for the API usability peer review.

Phase II: Review session. The duration of an API usability peer review session is 1.5 hours in a meeting room with all the roles synchronously present face-to-face (N.B. It is plausible to conduct such a session remotely, which may entail tradeoffs). The usability engineer consumes the first 5 minutes to enumerate the goals of the session to the reviewers. The feature owner consumes the next 10 minutes to present the user scenario and 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 feature owner walks the reviewers through the chosen scenario and associated tasks using the source code. Reviewers do not need to prepare any material prior to the review session. In this way, the feedback channel between the reviewers and feature owner is highly interactive. One aspect of usability that the walkthrough exposes is learnability — that is, how easy is it to understand a particular code module using the API. An initial encounter with a poorly designed API can leave a lasting impression of complexity and can even discourage users from ever adopting a particular technology [6]. One of the ways that users learn an API is through discovery of appropriate classes, methods, and parameters exposed by the API. For example, in Visual Studio, developers often use the IntelliSense feature [38] to learn and identify the appropriate members and parameters of a particular class or method to use. For each task, reviewers are prompted to consider the mismatches between the API designers' conceptualization and their own, poor choices of names, inadequate exposure of functions by the API to discover its underlying mechanics, and so forth. Reviewers are not experts or analysts trained in identifying API usability defects. The usability engineer anchors, leverages, and converges the feedback by the reviewers toward API usability issues and steers away from feedback related to

say, performance and architecture, which is more appropriate for technical API peer reviews.

The review process is a group-based walkthrough of the source code. The review is driven by probing reviewers around three questions: (1) What do you think the next step is; (2) What do you think this code module does; and (3) Does the implementation of the code module using the API make sense? The first question tries to identify mismatches between the reviewers' expectations and intentions of the API design, leading to an understanding of the end users' mental model. The second question probes the reviewers in thinking about what a code module is doing. This typically leads to a discussion with the feature owner on how difficult the code module is to understand. Once the reviewers understand what the code module is doing, the third question solicits feedback on implementation details. These think-aloud questions elicit responses from reviewers that eventually answer the key questions the feature owner defined in Phase I. Follow-up questions to the above-specified three questions generally tend to probe the rationale for reviewers' feedback (e.g., why do you think the code doesn't make sense) and possible suggestions (e.g. can you suggest a more discoverable method name).

Phase III: Identification of usability defects. The usability engineer and feature owner (feature area manager is optional) meet for one hour after the API usability peer review (typically the same or next business day). The raw notes recorded by the feature area manager and the usability engineer are used to identify actionable usability defects.

Example of a session

Here, we give a hypothetical yet typical example based on our experience of conducting an API usability peer review across the three phases. In the hypothetical example below, the API usability peer review refers to the System.Net.Communication.Sockets namespace, which is an API feature that allows developers to specify customized control on network communication using sockets.

Phase I is initiated by Ali, who is the feature owner of the namespace to be evaluated. He requests an API usability peer review by sending an email on Monday to Dan, who is the usability engineer responsible for that namespace. Dan sets up a 1-hour meeting with Ali on Tuesday morning. During the meeting, Ali describes two scenarios on which he wishes to get feedback: (1) defining socket-based messages, and (2) configuring network persistence. Dan feels that the two scenarios are quite large and cannot be accommodated in one session. Ali acknowledges Dan's concern and decides it's best to focus on the higher priority scenario of defining socket-based messages.

Dan prompts Ali to think about questions he would like to get answered from the API usability peer review. Ali replies that he wants to know how intuitive the sockets interface is for developers. Dan works with Ali to decompose this general question. After 10 minutes of deliberation, they both converge on three questions for the API usability peer review, one of them being the following, "How easy is it to define a socket-based message for routing to a central server?" (N.B. We used the word "intuitive" deliberately to signify how product teams think about usability.)

25 minutes into the meeting, Dan asks Ali about the source code to be used during the API usability peer review. Dan fires up Visual Studio and opens a "Hello World" sample application that implements basic socket-based messages. For about 15 minutes, Ali goes through the sample so that Dan has a rough understanding of the source code.

During the last 20 minutes of the Phase I meeting, Dan asks Ali to specify criteria for reviewers. Ali says the reviewers should have at least two years of C# experience and have at least six months' experience with the prior version of the .NET Framework. Dan wonders whether it would help to have at least two of the reviewers who have used the System.Net.Communication.Sockets namespace in the prior version of the .NET framework. Ali says that's a fantastic idea, as the API usability peer review will probably result in different feedback from reviewers who are new to the namespace and have used a prior version.

At the end of the meeting, Dan and Ali decide that the API usability peer review can be scheduled for Friday. Afterward, Dan sends an email containing a summary of the meeting to Ali with a copy to Melissa, the feature area manager (Ali's manager). Melissa is now aware of the API usability peer review and adds the session to her calendar. On Tuesday afternoon, Dan coordinates with Ali over email to recruit the reviewers. Dan sends an email to a pool of organizational peers who have expressed interest in being reviewers for different API features. Ali sends an email to his immediate colleagues in sister teams who are owners of other API features not related to Ali's feature. By Wednesday, Dan has confirmed three reviewers and Ali has confirmed one reviewer. On Thursday, Dan and Ali both send reminders to their respective reviewers to attend the API usability peer review session the next day.

Phase II is kicked off on Friday at 2PM when Ali, Dan, Melissa, and the four reviewers have all arrived in the conference room. Dan thanks the reviewers for taking out time to attend, introduces everyone, and gives an overview of what to expect in the session. Ali then takes the floor, fires up PowerPoint, and spends the next 10 minutes going through his two-slide presentation. The first slide is an overview of what the prior .NET Framework version provided in terms of network communication. The second slide is a process diagram showing information flow for the network communication scenario of using socket-based messages in the current version. After explaining the two slides, Ali proceeds to the whiteboard to illustrate the API feature under review.

At this point, Huang (one of the reviewers) asks Ali to explain the API feature in more detail. Ali responds, to which Huang asks whether or not the feature can support

distributed transactions. Ali, who is excited about answering Huang's question in detail, begins to vigorously describe the feature specification, on which he worked for months. Dan quickly realizes that the review session is turning into a tutorial. He politely interrupts Ali by saying: "The feature detail will come out during the review, but let's focus on starting the review itself so the product team gets the feedback they want." After an awkward pause, Dan asserts that providing too much detail upfront can bias the reviewer feedback. Everyone agrees, and 20 minutes into the session, the review starts.

Ali begins to describe a core task of defining a ReceiveMessage class using sockets. Before Ali proceeds to the definition mechanics, Dan intervenes: "Let's stop here, and ask the reviewers what they expect to do next." The next 10 minutes are spent on discussing what reviewers would expect to do next. After reaching a reasonable conclusion, Ali proceeds to show the first code snippet.

One of the method names stands out, and Isaac (another reviewer) immediately raises the issue: "In ReceiveMessage, why is the method called GetMsgTx?" Dan asks other reviewers if the method name makes sense. No one responds. Dan prompts Prakash and Victor to chime in, as they have been relatively quiet so far. Both reviewers come to the conclusion that the method name should avoid abbreviations and should explicitly spell out what the method is doing. Ali agrees. Dan checks with Melissa to make sure she noted the feedback.

Ali now drills into the method details. He asks the reviewers if the implementation makes sense. Huang and Prakash feel that one of the methods should be concrete. Ali disagrees, and gives the rationale that the method is abstract to allow flexibility in its configuration. Isaac jumps in and strongly expresses his opinion that an abstract method goes again design guidelines. Ali acknowledges Isaac's concern, but defends his choice as a tradeoff for providing flexibility. Isaac feels slighted and asserts that the guidelines are in place exactly for the reason of not violating them. At this point, Dan sees a conflict brewing. He states both side of the argument, and asks Isaac to suggest another workaround without violating the design guidelines. Isaac thinks for a moment and remembers a similar situation from his prior development experience. Isaac relates that experience. Melissa thinks Isaac's suggestion is a possibility that Ali can investigate offline. Dan ensures that Melissa records this feedback and the review goes on.

After the session ends at 3:30 when the reviewers leave, Dan asks Melissa and Ali to stay for identifying usability defects (Phase III) as they might lose important details over the weekend. As Ali is finalizing a list of issues, Dan prompts Melissa to specify any additional context, such as feasibility of the solution to address a particular usability defect and associated priority. As the post-mortem meeting progresses, Dan ensures that all the actionable feedback is captured by Ali as usability defects.

OVERVIEW OF STUDY

Method and data set

We conducted nine API usability peer reviews and three API usability tests in the lab during a six-month period. A total of 24 users were recruited across the three API usability tests (yielding an average of eight users per test). The duration of each API usability test session per user was approximately three hours, thus yielding 24 hours of user observation data per API usability test. In contrast, a *group* of reviewers were recruited for each API usability peer review session, thus yielding 1.5 hours of user observation data per API usability peer review.

The API usability peer reviews and API usability tests were conducted on part of Microsoft's .NET Framework APIs, though each API usability peer review and API usability test examined different API features. The same usability engineer conducted the API usability peer reviews and API usability tests. The API usability tests were conducted using standard lab methodology within Microsoft. Participants were recruited who had registered with the Microsoft Usability Research website (http://microsoft.com/usability). In compensation for their time, participants were given two vouchers each redeemable for a software or hardware item at the Microsoft Store, such as an Xbox game or a Microsoft keyboard. During an API usability test, users completed a series of programming tasks. All participants' screens using video were captured as well as their comments during the test. The video data was analyzed and usability defects were identified (see [36] for an example of the method used for an API usability test). In API usability tests, users actually wrote code using the tested APIs whereas in API usability peer reviews, users walked through the code.

The outcome of conducting both API usability peer reviews and API usability tests was a set of usability defects. The usability defects were problems or breakdowns that were observed during each session. Each usability defect was filed in the product team's bug tracking system (in our case, Team Foundation Server [37]). Henceforth, we refer to usability defects as usability bugs. Once the bugs were recorded, they were triaged by the product team.

The triaging process involves deliberating on each bug, assigning it appropriately to the product team member responsible for addressing the bug, and then closing the bug once it has been addressed. There are two important decisions in the triaging process that is of interest to us in this paper. First, each bug is classified into an issue type, summarized in Table 1.

Code defect bugs uncover problems in the underlying workings of the software code. Design change bugs represent major requests to introduce new design features or design modifications that require significant development and test effort to implement. Design issue bugs represent problems or tradeoffs in the current API design, which may or may not be addressed (if they are going to be addressed,

they are changed to a code defect or design change). Documentation bugs represent clarification in the accompanying API feature documentation or documentation that needs to be added. Sample bugs represent code defects specific to the code sample that is associated with the API feature. Work item bugs represent status tracking items for all other issue types so that all bugs are addressed in a timely manner.

Issue type	Definition
Code defect	Problem in underlying API that can be fixed by modifying code
Design change	Major design modification, requiring significant development and test effort
Design issue	Design problem or tradeoff, which may or may not need addressing
Documentation	Clarification in or missing API documentation
Sample	Code defect specific to API feature sample
Work item	Used for status tracking of bugs for other issue types

Table 1. Six possible issue types for bugs.

Second, each bug is assigned an importance via priority and severity, summarized in Table 2. Priority is determined by considering how urgent the bug needs to be fixed. Severity is determined by considering the impact of the bug to customers (e.g., bug resulting in system crash is S1). Depending on the bug, priority and severity may or may not mutually inform each other.

Priority	
Rating	*Definition*
P0	Bug should to be fixed immediately
P1	Bug should be fixed in next code check-in
P2	Bug should be fixed in current milestone
P3	Bug should be fixed in next milestone
Severity	
Rating	*Definition*
S1	Bug is a major functionality problem
S2	Bug is a moderate functionality problem
S3	Bug is a minor functionality problem
S4	Unsure if bug has any significant impact

Table 2. Four possible priorities and severities for bugs.

After a bug has been deliberated on and a resolution reached, the resolution is verified. Once the resolution is verified, the bug is closed. A closed bug is resolved by specifying a resolution type, summarized in Table 3.

Resolution	Definition
Fixed	Bug has been addressed through change
By design	Bug is an informed design decision
Duplicate	Bug has already been identified
Won't fix	Bug will not be addressed

Table 3. Four possible resolutions for bugs.

Research questions

The study presented in this paper answers the following two research questions.

- RQ1: Are API usability peer reviews useful in identifying usability bugs that are relevant to the product team?
- RQ2: How do usability bugs identified from API usability peer reviews compare with usability bugs identified from API usability tests?

Regarding RQ1, if API usability peer reviews are useful, they should identify usability bugs that are new to the product team. Further, these usability bugs should have a positive impact on the underlying API.

Decomposing RQ2, our analysis compares usability bugs between API usability peer reviews and API usability tests across four attributes:

- RQ 2(a): Quantity of bugs. Do the methods result in different number of bugs?
- RQ 2(b): Types of bugs. Are the methods sensitive to different types of bugs?
- RQ 2(c): Importance of bugs. Do the methods identify bugs of different priority and severity?
- RQ 2(d): Resolution of bugs. Do the methods differ in how the bugs are resolved?

RESULTS

RQ1: Are API usability peer reviews useful?

From the nine API usability peer reviews, a total of 51 usability bugs were identified. Of these 51 usability bugs, 34 have been resolved while the remaining 17 are being triaged by the product team. Of the 34 resolved bugs, seven (21%) have been resolved as "Fixed", six (18%) as "By design", 14 (41%) as "Won't fix", and seven (21%) as "Duplicate".

RQ2: How do usability bugs compare between API usability peer reviews and API usability tests?

RQ 2(a): Quantity of bugs

From the three API usability tests, a total of 41 usability bugs were identified. Figure 2 shows a box plot of the number of bugs identified by each method. There was a significant difference ($t(10) = 2.78$, $p < 0.05$) in the mean number of bugs identified between API usability tests (Mean = 13.7, S.D. = 3.7) and API usability peer reviews (Mean = 5.7, S.D. = 3.6) per session.

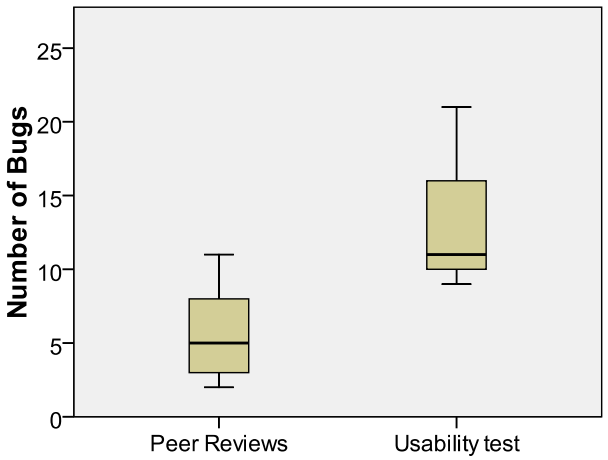

Figure 2. Difference in number of bugs across methods.

RQ 2(b): Types of bugs

For both methods, we calculated the distribution of bugs (N = 92) across the six issue types. Figure 3 shows the differences in percentages.

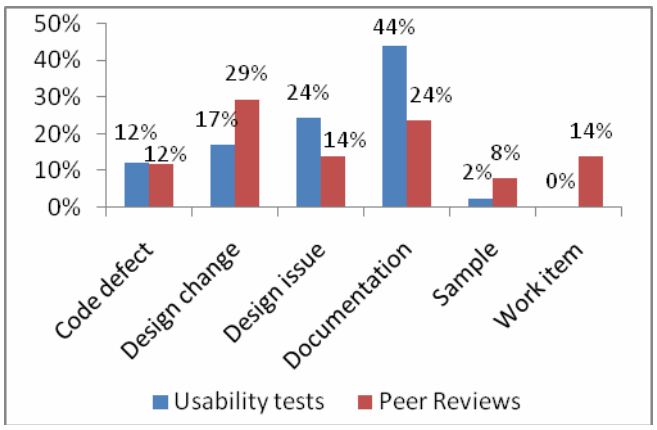

Figure 3. Distribution of bugs across bug types.

RQ 2(c): Importance of bugs

For both methods, we calculated the distribution of bugs (N = 92) across priority (P) and severity (S). Figure 4 shows the differences in percentages.

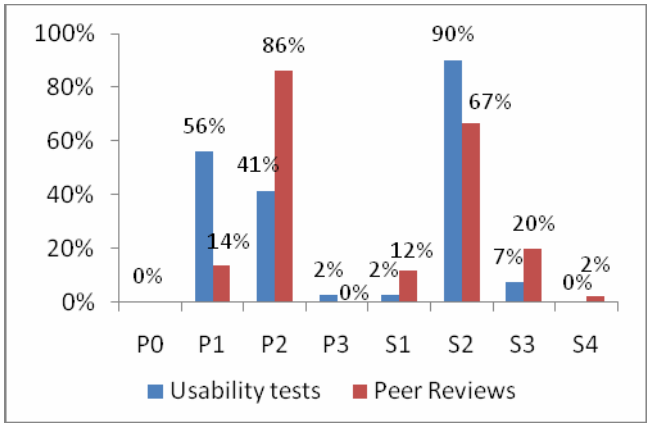

Figure 4. Distribution of bugs across bug priority and severity.

RQ 2(d): Resolution of bugs

For both methods, we calculated the distribution of resolved bugs (N = 55) across resolution type. Figure 5 shows the differences in percentages.

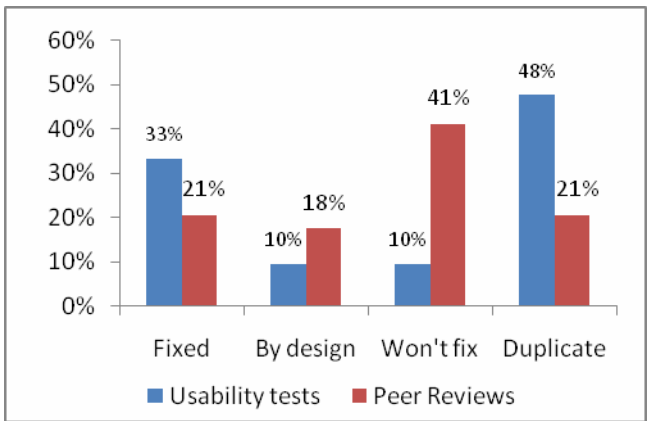

Figure 5. Distribution of bugs across bug resolution.

DISCUSSION

Discussion of results

Regarding the first research question of whether or not API usability peer reviews are useful, our results show that 59% of the usability bugs (Fixed and Won't fix) were new to the product team (i.e., bugs that were not identified before). The RQ1 result also shows that 21% of the bugs were fixed, demonstrating that API usability peer reviews had a positive impact on the API. Given our goal for usability research — find new usability defects to enhance the usability of the API — this result indicates that these API usability peer reviews were a useful method.

Results from the second research question (RQ2) suggest some possible differences between API usability peer reviews and API usability tests. Regarding RQ 2(a), API usability tests tend to identify more bugs per session than API usability peer reviews. As mentioned in the overview above, each API usability test yielded 24 hours of user observation data (users writing code using API) whereas each API usability peer review yielded 1.5 hours of user observation data (users walking through API code). This additional time plus the in-depth interaction with the API in API usability tests could be expected to generate more usability bugs. This result was largely expected.

Results from RQ 2(b) suggest that the distribution of bugs across bug issue types may be different across the methods. In API usability tests, users were asked to read preliminary API documentation and then perform actual programming tasks (users could refer to documentation during the tasks). This likely explains why API usability tests uncovered more documentation issues (44% of bugs for API usability tests vs. 24% for API usability peer reviews). The actual user interaction with the API during API usability tests possibly contributed to revealing more design issues (24% of bugs for usability tests vs. 14% for peer reviews), as tradeoffs in design are more likely to be uncovered through

hands-on experimentation with code (e.g., doing a programming task in two different ways, and then comparing and contrasting the two ways). On the other hand, API usability peer reviews are more conceptual and less procedural in nature. During the face-to-face session, reviewers engage in a discourse with the feature owner and thus are more likely to expose the feature owner's rationale for designing the API feature in a particular way. This can lead to new design insights and ideas, and may explain a higher proportion of design change issues (29% of the bugs for API usability peer reviews vs. 17% for API usability tests). The discourse between reviewers and feature owner can also prompt the feature owners to record follow-up items on bugs that are already being addressed so as to ensure they get tracked. Given that API usability peer reviews have quick execution time (actual session is 1.5 hours and post-mortem meeting is one hour), feature owners find it helpful to maintain a tracking record by specifying certain bugs as work items only revealed by API usability peer reviews (14%).

Results from RQ 2(c) suggest that the distribution of bugs across priority and severity may be different across the methods. Specifically, API usability tests identified higher priority bugs (56% P1 bugs) than API usability peer reviews (86% P2 bugs). This may be explained by the evidence that API usability tests identified more documentation issues, which are more feasible to fix than other issue types that may require architectural changes in the underlying API. Another possible explanation is that API usability tests have higher predictive and content validity. In API usability tests, users represent the profiles of target customers more so than users in API usability peer reviews. Further, users in API usability tests are actually programming using the API, which provides a real-world context. Thus, the higher predictive and content validity may explain higher confidence of feature owners in specifying bugs from API usability tests as more important that need to be fixed sooner.

Results from RQ 2(d) suggest that the distribution of bugs across resolution may be different across the methods. The API usability tests and API usability peer reviews reported in this paper were conducted late in the development cycle (i.e., close to product release). Thus, API usability tests were requested by the product team to validate existing bugs on API features that partially had been tested before. That is, product teams were largely looking for additional user data and context on already known bugs, which could explain the higher proportion of "Duplicate" bugs (48% of the bugs from API usability tests vs. 21% from API usability peer reviews). On the other hand, API usability peer reviews were conducted on API features not tested before, leading to lesser "Duplicate" bugs. Fixing new bugs late in the development cycle carried high regression risk and development/test cost, which may explain the higher proportion of "Won't fix" bugs (41% for API usability peer reviews versus 10% for API usability tests).

Implications

Our motivation to innovate a usability inspection method for APIs was a pragmatic one. We had an extremely large API to evaluate with limited resources. API usability peer reviews allowed us to dramatically increase the usability coverage of the .NET Framework APIs we supported.

API usability tests have almost a 2.5x productivity advantage over API usability peer reviews (ratio of the mean number of bugs *per test/session iteration*) despite approximately a 16x advantage in user observation data (ratio of the user observation time). In other words, this implies that on average, each API usability test is roughly equivalent to 16 API usability peer reviews (N.B. Note that *per unit of time*, API usability peer reviews are more efficient and thus have a productivity advantage over API usability tests). From a usability practitioner's perspective, API usability peer reviews are significantly more efficient than API usability tests and thus provide a more scalable method for evaluating APIs.

Based on our preliminary analysis, API usability peer reviews have been shown to be an effective method in identifying new usability defects. In comparing API usability peer reviews with API usability tests, we found evidence that suggests the methods are complementary (i.e., sensitivity to different issue types):

- API usability tests expose design issues related to actually using an API and related resources including documentation.
- API usability peer reviews expose the design rationale of an API and uncover conceptual flaws in its design.

The complementary nature of the methods is consistent with Greenberg and Buxton's [17] argument. While API usability peer reviews may identify some of the same issues as API usability tests, the types of issues found are more likely to be addressed earlier in the design and development process. From that perspective, API usability peer reviews are similar to other walkthrough methods (see discussion on "when to use inspection methods" in [27, pp. 18-19]). From our anecdotal experience with socializing the method with product teams, we expect that the usability bugs from API usability peer reviews conducted earlier in the development cycle with formative API features (raw specifications or even stubbed out code modules) will likely have a higher fix rate than reflected in our results. Table 4 summaries the advantages and disadvantages of both methods, highlighting our results.

One of the implications of API usability peer reviews is how they are adopted in software organizations that already have an institutionalized practice of conducting API usability tests. In our experience, the collaborative practice of socially reviewing APIs through API usability peer reviews aligns well with existing software design culture. At Microsoft, product teams could relate to API usability peer reviews based on their prior experience with collaborative software engineering practices through agile

software development such as Scrum [34]. A number of Microsoft product teams have adopted the concept of feature crews that emphasize frequent and iterative communication between the multiple disciplines involved in designing, implementing, testing, and releasing products [12]. This enables teams to obtain user feedback on a regular basis. Streamlining agile-based practices to evaluate API usability using API usability peer reviews was a smooth transition for product teams at Microsoft.

	API usability	API usability tests
Advantages	▪ Lower cost ▪ Shorter execution time ▪ More sensitive to design changes and work items	▪ Identifies more usability defects ▪ More sensitive to design and documentation issues ▪ Identifies higher priority defects
Disadvantages	▪ Identifies lesser usability defects ▪ Less sensitive to design and documentation issues	▪ Higher cost ▪ Longer execution time ▪ Less sensitive to design changes and work items

Table 4. Comparative summary of API usability peer reviews and API usability tests.

Threats to validity

As a function of conducting our research in a naturalistic setting, we were unable to control for certain factors. For example, the API usability tests and API usability peer reviews were not conducted on exactly the same API features. Practically, the identification of API features to be evaluated was largely dictated by improving usability coverage across the API, thus implying that the evaluation would favor usability testing of distinct API features not tested before. We also could not control the development stage during which the API usability tests and API usability peer reviews were conducted. Indeed, both these factors are likely to have influenced the assignment of bug issue types across different features and by extension their resolution and the assignment of bug priorities and severities across different development milestones.

CONCLUSION AND FUTURE WORK

It is important to the field of HCI to develop and compare methods by which we assess the usability of technological artifacts and by extension APIs [7, 8]. The Web 2.0 phenomenon has contributed to an increased number of public APIs that software organizations expose to users as a service for developing customized applications (e.g. Google Maps API [14], iPhone SDK [20]). In this paper, we: (a) describe a usability inspection method that software organizations can scale to evaluate the usability of a large number of APIs with few usability resources; and (b) compare our proposed method with an empirical method to understand how they might complement each other. We believe that usability practitioners and researchers will find our contribution useful as they adapt various agile methods and approaches for evaluating API usability in order to improve the overall end user development experience.

As we are ramping up to plan for the next release of the .NET Framework, our immediate future work will incorporate API usability peer reviews and API usability tests earlier in, and throughout, the development cycle. This will give us flexibility in analyzing both methods in a relatively controlled manner. We also want to do a content analysis of the usability defects from both methods. Such an analysis would provide qualitative data to further explain some of the quantitative trends reported in this paper. We see these analyses defining our future research trajectory as we move forward in making API usability peer reviews a more pervasive organizational practice at Microsoft.

ACKNOWLEDGEMENTS

We thank Bob Schmidt and Karen Swanson for advising the development of the method. We are grateful to John Wyss and Jurgen Willis for their sponsorship. Elizabeth Buie and John Daughtry helped to revise the paper. The content presented in this paper represents the sole opinions of the authors and not of Microsoft.

REFERENCES

1. Ackerman, A.F., Buchwald, L.S., and Lewski, F.H. Software inspections: An effective verification process. *IEEE Software 6*, 3 (May 1989), 31-36.
2. Bell, B. *Using programming walkthroughs to design a visual language*. Ph.D. dissertation, University of Colorado, 1992.
3. Bell, B., Citrin, W., Lewis, C., Rieman, J., Wilde, N., and Zorn, B. The programming walkthrough: A structured method for assessing the writability of programming languages. *Software Practice and Experience 24*, 1 (January 1994), 1-25.
4. 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.
5. Cuomo, D.L. and Bowen, C.D. Understanding usability issues addressed by three user-system interface evaluation techniques. *Interacting with Computers 6*, 1 (1994), 86-108.
6. Cwalina, K. and Abrams, B. *Framework design guidelines*. Addison-Wesley, 2005.
7. Daughtry, J.M., Farooq, U., Stylos, J., and Myers, B.A. API usability: CHI'09 Special Interest Group. *Proc. CHI 2009*, ACM Press (2009), 2771-2774.
8. Daughtry, J.M., Farooq, U., Myers, B.A, and Stylos, J. API usability: Report on Special Interest Group at CHI. *Software Engineering Notes 34*, 4 (July 2009), 27-29.
9. Desurvire, H.W. Faster, Cheaper!! Are Usability Inspection Methods as Effective as Empirical Testing?

In Nielsen, J. and Mack, R.L. *Usability inspection methods* (Ed). John Wiley & Sons, 1994, 173-201.

10. Fagan, M.E. Design and code inspection to reduce errors in program development. *IBM Systems Journal 15*, 3 (1976), 182-211.
11. Fagan, M.E. Advances in software inspection. *IEEE Transactions on Software Engineering 12*, 7 (July, 1986), 744-751.
12. Feature Crews: How Microsoft Does It. CodePlex: Open Source Community, 2007. http://www.codeplex.com/BranchingGuidance/Wiki/View.aspx?title=Feature%20Crews%3a%20How%20Microsoft%20Does%20It&referringTitle=Home.
13. Freedman, D. and Weinberg, G.M. *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*. New York: Dorset House, 1990.
14. Google Maps API. http://code.google.com/apis/maps/.
15. Gray, W.D. and Salzman, M.C. Damaged merchandise? A review of experiments that compare usability evaluation methods. *Human Computer Interaction 13*, 3 (1998) 203-261.
16. 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.
17. Greenberg, S. and Buxton, B. Usability evaluation considered harmful (some of the time). *Proc. CHI 2008*, ACM Press (2008), 111-120.
18. Hammond, N., Hinton, G., Barnard, P., MacLean, A., Long, J., and Whitefield, A. Evaluating the interface of a document processor: A comparison of expert judgment and user observation. *Proc. of IFIP INTERACT 1994*, Elsevier Science Publishers (1994), 725-729.
19. Henning, M. API Design Matters. *Communications of the ACM 52*, 5 (May 2009), 46-56.
20. iPhone Developer Program. http://developer.apple.com/iPhone/program.
21. Jeffries, R. and Desurvire, H. Usability testing vs. heuristic evaluation: Was there a contest? *ACM SIGCHI Bulletin 24*, 4 (October 1992), 39-41.
22. Jeffries, R., Miller, J.R., Wharton, C., and Uyeda, K.M. User Interface Evaluation in the Real World: A Comparison of Four Techniques. *Proc. CHI 1991*, ACM Press (1991), 119-124.
23. John, B.E. and Marks, S.J. Tracking the effectiveness of usability evaluation methods. *Behaviour & Information Technology 16*, 4/5 (1997), 188-202.
24. Kahn, M.J. and Prail, A. Formal usability inspections. In Nielsen, J. and Mack, R.L. *Usability inspection methods* (Ed). John Wiley & Sons, 1994, 141-171.
25. Karat, C-M., Campbell, R., and Fiegel, T. Comparison of empirical testing and walkthrough methods in user interface evaluation. *Proc. CHI 1992*, ACM Press (1992), 397-404.
26. Lewis, C., Polson, P.G., Wharton, C., and Rieman, J. Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces. *Proc. CHI 1990*, ACM Press (1990), 235-242.
27. Mack, R.L. and Nielsen, J. Executive summary. In Nielsen, J. and Mack, R.L. *Usability inspection methods* (Ed). John Wiley & Sons, 1994, 1-23.
28. Nielsen, J. and Mack, R.L. *Usability inspection methods* (Ed). John Wiley & Sons, 1994.
29. Nielsen, J. and Phillips, V. Estimating the relative usability of two interfaces. *Proc. CHI 1993*, ACM Press (1993), 214-221.
30. Olson, G.M., and Moran, T.P. Commentary on "Damaged Merchandise". *Human Computer Interaction 13,* 3 (1998), 263-323.
31. Rieman, J., Franzke, M., and Redmiles, D. Usability evaluation with the cognitive walkthrough. *Proc. CHI 1995*, ACM Press (1995), 387-388.
32. Russell, G.W. Experience with inspection in ultralarge-scale developments. *IEEE Software 8*, 1 (January 1991), 25-31.
33. Savage, P. User interface evaluation in an iterative design process: A comparison of three techniques. *Proc. CHI 1996*, ACM Press (1996), 307-308.
34. Schwaber, K. and Beedle, M. *Agile software development with SCRUM*. Prentice Hall, 2002.
35. Stevens, S.M. Intelligent interactive video simulation of a code inspection. *Communications of the ACM 32*, 7 (July, 1989), 832-843.
36. Stylos, J. and Clarke, S. Usability implications for requiring parameters in objects' constructors. *Proc. ICSE 2007*, ACM Press (2007), 529-539.
37. Team Foundation Server. http://msdn.microsoft.com/en-us/teamsystem/dd408382.aspx.
38. Visual Studio IntelliSense. http://msdn.microsoft.com/en-us/library/hcw1s69b(VS.71).aspx.
39. Vygotsky, L.S. Mind in society: The development of higher psychological processes. Harvard University Press, 1978.
40. Weller, E.F. Lessons from three years of inspection data. *IEEE Software 10*, 5 (September, 1993), 38-45.
41. 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).
42. Wiegers, K.E. *Peer reviews in software: A practical guide*. Addison-Wesley, 2002.