Informing API Design through Usability Studies of API Design Choices: A Research Abstract

Jeffrey Stylos
Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
jsstylos@cs.cmu.edu

Abstract

Using APIs is a common and often difficult task for developers. Successful API designs can guide users of an API and reduce their dependence on documentation; however API design is far from a science. Usability lab studies have been shown to be successful at improving the usability of specific APIs; however these are expensive and not always possible to run for every API. This paper describes an approach to generalize from studies of specific APIs to investigate the usability impact of design choices that commonly arise in the creation of APIs. Based on these results we will inform the design of many new APIs. A preliminary usability study of whether or not to require constructor parameters confirms our belief that the answer to common design decisions is not always obvious, and making the wrong choice can have a strong negative impact on usability for large groups of API users.

1. The Problem

There are more software libraries, frameworks, toolkits and application programming interfaces (APIs) than ever and new versions and new libraries are released every year. Applications have grown more dependent on these libraries, and figuring out how to use them presents several barriers for programmers [6].

Good API design can greatly reduce the time and difficulty it takes a developer to use an API, and can help avoid costly bugs in code that uses the API. However, different developers have different needs and learning styles, and there are only a few guidelines to help with API design [3].

Researchers at Microsoft have developed techniques for studying the usability of their new APIs, and analyses for interpreting the results that are inspired by the cognitive dimensions framework. By studying how programmers use APIs in a lab environment and analyzing the results in the context of the cognitive dimensions, they have successfully improved the usability of their new APIs [2].

A limitation of this technique is that it requires user testing each API, which is expensive for organization creating tens of thousands of APIs, and requires resources that smaller organization may not have.

2. Approach

Our approach is to compare the usability differences of specific API design decisions, while controlling for biases of any particular API. For example, in a case study, we evaluated an *object constructor* design choice: whether or not to require constructor parameters (instead of making them optional). The results of these studies inform the design of all APIs that might use these patterns, for different groups of programmers.

To study the usability of API design choices, we apply techniques from individual API usability studies [2] and also construct new techniques. In addition to using small realistic programming tasks with real APIs, we wrap and modify existing APIs to directly compare multiple conditions; generate "artificial" APIs to test specific circumstances, and control for experience with APIs and biases that come from specific real world domains. We also have programmers use Notepad to write the code they expect would solve a task to capture their expectations.

Based on experienced with specific API studies, we tailor usability tests for different groups of programmers, to accurately capture their typical tasks and work environment. We use the cognitive dimensions framework to analyze our results and understand not just what the problems were, but to understand what their root causes were, and what types of solutions might help.

Visual Languages and Human-Centric Computing (VL-HCC'06)
0-7695-2586-5/06 $20.00 © 2006 IEEE

3. A Case Study

`var foo = new Foo(barVal);`
`foo.Use();`
	`var foo = new Foo();`
	`foo.Bar = barVal;`
	`foo.Use();`

Required constructor call	"Create-set-call"

Figure 1. Constructor design options

As a case study of the types of techniques we propose [7], we studied the effects of requiring essential components of an object in the constructor (by not exposing a default constructor) versus providing a default constructor (potentially allowing the objects to be created and used invalidly by never specifying the essential components).

We hypothesized that required constructor parameters would guide consumers of the API toward correct usage of the object, creating APIs that were more "self-documenting."

To test this hypothesis, we designed a series of small tasks using the techniques from the previous section, and presented these to programmers in two usability studies. We were interested not just in the initial task of code creation, but also in getting a larger picture about all of the usability effects throughout the lifetime of the code.

We were surprised to find from our studies that contrary to our expectations programmers preferred and were more successful with APIs that did *not* require any constructor arguments. The reasons for this became clearer when we used the cognitive dimensions to analyze the results of our studies. One reason was required constructor parameters forced programmers to instantiate each of the parameter objects before they felt they could explore the constructed object using design-time tools like code completion. This caused a problem of premature commitment [5] in the APIs that used required constructor parameters.

Additionally, the programmers that were most concerned about object consistency, whom we assumed would most prefer required constructor parameters since they prevent inconsistent objects, found the guarantees of required constructors insufficient for their high-reliability needs. Instead, they preferred being able to explicitly set each parameter's value, since this allowed finer control of errors (since constructors cannot return error codes).

4. Proposed Research Direction

We propose to apply similar methodology to study other API design choices.

One related design choice is when to use the factory and builder design patterns [4] for objects. Design guideline references sometimes recommend the use of factories in APIs because of the flexibility they offer to API implementers and because of claims of increased readability [1]. However, based on our observations of programmers using specific APIs, and based on our constructor-parameter studies, we hypothesize that there is a strong discoverability cost of using factories instead of constructors that for some programmers could outweigh all of the benefits. We plan to design a usability study using the techniques we have developed to test this hypothesis.

There are many different design choices that API designers are faced with that could be usefully studied in this manner, including how to expose concurrency and asynchronous operations (via threads, events, callbacks or other mechanisms); how to decompose functionality into multiple classes; how and when to give error messages to consumers of an API; and numerous other choices.

To decide which API design choices would be most usefully studied, we plan to survey the most common and severe usability problems that programmers face in existing APIs. From these observations, we plan to identify usability problems caused by API design choices that have reasonable, possibly more usable, alternatives. The design choices that will be most useful to study are those that are sufficiently common (occurring in many real APIs), severe (causing non-negligible problems), and fixable (where we might be able to suggest a better alternative).

5. Acknowledgements

This work was partially supported by NSF grant IIS-0329090 and the EUSES Consortium via NSF grant ITR-0325273. Opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the NSF.

6. References

[1] Bloch, J.: Effective Java Programming Language Guide. Mountain View, CA, Sun Microsystems (2001)

[2] Clarke, S.: Measuring API usability. Dr. Dobbs Journal (2004) S6-S9

[3] Cwalina, K. and B. Abrams: Framework Design Guidelines, Addison-Wesley Professional (2005)

[4] Gamma, E., R. Helm, R. Johnson, J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Boston, MA, USA, Addison-Wesley. (1995)

[5] Green, T. R. G. and M. Petre: Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework., Journal of Visual Languages and Computing 7(2) (1996) 131-174

[6] Ko, A. J., B. A. Myers, H. H. Aung: Six Learning Barriers in End-User Programming Sys-tems. VL/HCC, 2004 IEEE Symposium on. (2004)

[7] Stylos. J., S. Clarke, B. A. Myers: Studying the Usability of API Design Choices. Submitted for Publication. (2006)