Measuring API Usability

Usability of software tools impacts developer efficiency

Steven Clarke

The Visual Studio usability group at Microsoft, of which I'm a part, is responsible for helping design and build the user experience for developers using Visual Studio and the .NET platform. Much of our efforts at improving the developer experience have focused on improving the usability of the Visual Studio development tool itself. For example, we run extensive empirical studies in usability labs focusing on particular features of Visual Studio (such as the debugger), gathering data to help feature teams improve their designs. In this way, we can understand how best to present complex information, such as the structure of application data, to support developers better when they are debugging code.

But while improving the design of Visual Studio is a necessary component of improving the developer experience, it is not sufficient. The development tool is only one tool that developers routinely use when building applications. The other major tools are the programming language they are coding in (C++, C#, VB, and so on) and any class libraries or APIs that they are coding against (MFC, .NET Frameworks, and the like). The usability of the languages and libraries that developers use have a significant impact on their ability to successfully complete a set of development tasks. In this article, I describe some of the techniques we use to design and evaluate the usability of the APIs that ship with .NET, and discuss ways in which you can use them for APIs you are designing.

Steven is a usability engineer at Microsoft, working on Visual C++, the .NET Frameworks, and the WinFX libraries. Prior to joining Microsoft, he was a developer at Motorola, building development tools for Smartcard applications. Steven can be contacted at stevencl@microsoft.com.

Usability Applied to APIs

Most of us know what usability means when applied to software with graphical user interfaces (GUIs)— developer tools, word processors, or e-mail clients. We expect these tools to let us perform a given set of tasks and to work the way we expect them to. If I'm using a visual debugger, for example, I expect to be able to examine the state of a variable at some point during the execution of my code. I expect to be able to do this by using some debugger tool window that shows a list of variables that are currently in scope and selecting the variable that I want to examine. Or, if I am checking my e-mail, I want to see the new, unread e-mail that has arrived since I last read my e-mail. I expect to be able to do this by opening the client and having the new e-mail displayed in a way that distinguishes it from the old e-mail that I have read. If any of the products that we use fail to work in the way that we expect them to, we generally label this as poor design or usability.

There can be many causes for poor product usability or design; for example, users may not be aware of the actions that they can take with a product. Likewise, they may not be able to predict the result of performing some action and hence might be reluctant or less inclined to perform that action for fear of the potentially negative consequences that might ensue. Or they misinterpret the actions that they can take and the results of those actions.

Psychologists use the term "affordances" to refer to the actions that a user can take on some object. However, just because an object "affords" some action, it doesn't mean that the user will realize that they can take that action— users have to be able to perceive the affordances exposed by some object. Many usability breakdowns that occur in GUIs are due to affordances not being perceived by users. For example, for some users, the Start button in Windows XP is not perceived to afford the action of shutting the computer down.

Likewise, APIs expose affordances. Every API has a set of actions that it can perform. Therefore, usability problems can exist in an API that are related to users not perceiving the affordances the API supports. For example, an API for performing file I/O might afford creating a new file, writing to a file, reading from a file, deleting a file, copying a file, and moving a file; see Example 1. However, developers browsing the public interface of this API might not perceive the complete set of affordances that this API offers. In particular, they might not realize that they can use this API to move a file from one location to another by copying the file, then deleting the original. Or they may not realize that the constructor creates a new file, if the path specified in the constructor does not already point to an existing file.

Designing Usable APIs

One of the best ways to design usable GUIs or APIs is to follow a user-centered-design approach. User-centered-design encompasses understanding both your users and the way that they work in the scenarios the product is designed to support. Application implementation details do not drive the design of the user interface and should not be represented in the interface design. Instead, the design should reflect the way that users expect to work.

Designing usable APIs is similar to designing usable GUIs. You must make sure that you understand the characteristics of users and how those characteristics impact the way they expect the API to work. Following a scenario-based design approach ensures that the API reflects the tasks that users want to perform, instead of reflecting the implementation details of the API.