Separating the Wheat from the Chaff in Internet-Mediated User Feedback

David M. Hilbert David F. Redmiles
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1 714 824 3100
{dhilbert,redmiles}@ics.uci.edu

ABSTRACT

The Internet enables cheap, rapid, and large-scale distribution of software for evaluation purposes. It also presents hitherto unprecedented, and currently underutilized, opportunities for increasing user-developer communication in software development. For instance, the Internet can be used as a medium for collecting "direct" user feedback in the form of subjective user reports, as well as "indirect" feedback in the form of automatically-captured data about application and user behavior. Both of these practices, however, face a number of challenges that can be summarized in the following statement: there is more feedback to be collected — ranging in quality from useful to useless — than there is time and resources to sift through and act upon the meaningful parts. This paper describes an Internet-based approach for capturing user feedback — both "direct" and "indirect" — that attempts to address this problem by focusing feedback collection based on the notion of "usage expectations" in the development process.

Keywords

Internet-mediated user-developer communication, user feedback, remote usability evaluation, expectation-driven event monitoring

INTRODUCTION

The Internet and World-Wide-Web make it possible to rapidly distribute prototypes and beta releases to large numbers of users at low cost. In principle, the Internet could become a large-scale test-bed for gathering "direct" user feedback in the form of subjective user reports, and "indirect" user feedback in the form of automatically-captured data about application and user behavior. For example, prototypes and beta releases are now commonly used in conjunction with Internet-based groupware technologies, such as news groups and e-mail lists, in order to collect "direct" user feedback. Companies like Aqueduct Software and Full Circle Software are now marketing Internet-based mechanisms for automatically collecting "indirect" user feedback, in the form of system and user performance data, in addition to subjective user reports. However, in practice, each of these approaches faces critical challenges that make it difficult to separate important feedback from unimportant feedback.

DIRECT FEEDBACK

Direct feedback is information that is gathered directly from users based on their subjective reports. Methods for collecting direct feedback range from relatively free-form discussion forums (e.g., news groups and email lists), to user-identified "critical incidents" [3], to questionnaires and interviews, to more ethnographically inspired techniques involving joint observer-participant interpretation of behavior in context. A key benefit of direct feedback is its ability to capture aspects of users' needs, desires, thought processes, and subjective experiences that are difficult to obtain using the indirect approaches described below.

When the Internet is used as a means for collecting this sort of feedback, however, there are a number of challenges that arise. Some of these challenges become clear when considering the case of beta testers who take it upon themselves to report usability issues in addition to software bugs.

First, the incentive structure for providing such feedback is typically unfavorable to users. Many problems go unreported because users are more interested in getting their work done than in paying the price of problem reporting while developers receive most of the benefit. Typically only the most obvious or unrecoverable errors are reported.

Second, there is often a paradoxical relationship between users' performance with respect to a particular application and their subjective ratings of its usability. Numerous usability professionals have observed this phenomenon. Users who perform well in usability evaluations often report problems with the interface even though these problems apparently did not affect the user's ability to complete tasks. When asked for a justification, these users will typically say something to the effect: "Well, I had an easy time, but I think other people would have been confused." Sometimes these users correctly anticipate problems encountered by other, less seasoned, users, however, this sort of feedback is based

on speculation and frequently turns out to be unfounded. On the other hand, users who encounter great difficulties using a particular interface will report that the interface is well designed and easy to use. When confronted with the discrepancy between their subjective report and observed behavior, these users will typically say something to the effect: "People with more experience would probably have had an easier time," or "I always have more trouble than average with this sort of thing."[1] As a result, potentially important feedback from users having difficulty will fail to be reported while potentially misleading or unfounded feedback from users having no difficulties will be reported.

Nevertheless, beta tests do appear to offer good opportunities for collecting useful usability information. Smilowitz and colleagues showed that beta testers who were asked to record usability problems as they arose in normal use identified almost the same number of significant usability problems as identified in more formal lab evaluations of the same software [6]. (A later case study performed by Hartson and associates, using a remote data collection technique, also appeared to support these results [3]). However, while the number of usability problems identified in the lab and beta test conditions was roughly equal, the number of common problems identified by both was rather small. In summarizing their results, Smilowitz and colleagues offered the following as one possible explanation:

> Another reason for this finding may have to do with the individual identifying the problems. In the lab test two observers with experience with the software identified and recorded the problems. In some cases, the users were not aware they were incorrectly using the tool or understanding how the tool worked. If the same is true of the beta testers, some severe problems may have been missed because the testers were not aware they were encountering a problem, and therefore did not record the problem [6].

Since users are not always aware of developers' expectations about appropriate usage, they can behave in ways that blatantly violate developers' expectations without being aware of it. As a result, mismatches between developers' expectations and actual usage may go undetected, resulting in ongoing usability issues.

Another important limitation identified by Smilowitz and colleagues is that the data reported in the beta test condition lacked details regarding user performance and the frequency of occurrences of problems.

INDIRECT FEEDBACK

In contrast, *indirect feedback* is information that is gathered by observing users as they interact with applications. Methods for collecting indirect feedback range from direct observation, to audio and video recording and conferencing, to automated software monitoring, to psychophysical event monitoring (e.g., galvanic skin response and electroencephalograms). A key benefit of indirect feedback is that it tends to capture detailed, "objective" information about user and application behavior (e.g., counts, sequence, and timing of actions) that are difficult to obtain using the direct feedback techniques described above.

When the Internet is used as a means for collecting this sort of feedback, however, there are a number of challenges that arise. Some of these challenges become clear when considering various approaches that have been proposed for performing remote usability evaluations.

Some investigators are beginning to investigate techniques based on audio and video conferencing and remote "application sharing" technologies. Unfortunately, while leveraging the Internet to overcome geographical barriers, these techniques fail to exploit the enormous potential afforded by the Internet for capturing user feedback on a large scale. This is because a large amount of data is generated per user, and because observers are typically required to observe and interact with users on a one-on-one basis.

An alternative to such techniques involves automatically capturing information about application and user behavior. A number of application instrumentation and event monitoring techniques have been proposed for this purpose. However, most of these approaches suffer from some combination of the following problems: (1) Low signal-to-noise ratio: too much data is collected, and it is difficult to separate the useful data from the noise. (2) Missing context: information critical in interpreting the meaning of user actions is often missing from the event stream, making post-hoc analysis uncertain and challenging at best. (3) Lack of appropriate abstraction: events are captured and analyzed at the window system level, or just slightly above, making it difficult to analyze user behaviors at higher levels of abstraction. (4) One-way communication: data flows from users to developers who must make inferences about what is going on. No "dialogue" is established to facilitate mutual understanding. (5) Batch orientation: hypothesis formation and analysis is performed after large amounts of (potentially irrelevant) data have been collected. It is difficult to tailor data collection based on hypotheses, or for hypotheses to be analyzed and action taken while users are engaged.

A NOVEL APPROACH

This paper presents a semi-automated, Internet-mediated approach to user feedback collection that attempts to address these issues, making it possible to capture higher quality feedback on a much larger scale than is currently possible. This approach is based on the notion of monitoring "usage expectations" as users interact with software applications.

Expectations in the Development Process

When developers design systems, they have numerous expectations about how those systems will be used. We call these usage expectations [1]. When developers' expectations don't match actual usage, various problems may ensue that affect the usability and ultimate utility of the application in question.

1.These examples were taken from a discussion that appeared in a closed, private discussion group, available only to practicing professionals in the usability and human-computer interaction community.

Developers' expectations are based on their knowledge of the requirements, past experience in developing systems, knowledge of the domain, knowledge of the specific tasks and work environments of users, and past experience in using applications themselves. Some of these expectations are explicitly represented — for example, those specified in requirements, use cases, or cognitive walkthroughs. Some are implicit — including assumptions about usage that are encoded in screen layout, key assignments, program structure, and user interface libraries.

For instance, implicit in the layout of most data entry forms is the expectation that users will complete them from top to bottom, with only minor variation. Also, menu and toolbars are typically laid out based on expectations about frequency of use. Such expectations are typically not represented explicitly, and as a result, fail to be tested adequately.

Detecting and resolving mismatches between developers' expectations and actual usage is important in improving the fit between application design and use. Once mismatches are detected, they may be corrected in one of two ways. Developers may change their expectations about usage to better match actual use, thus refining system requirements and eventually making a more usable system. For example, features that were expected to be used rarely, but are used often in practice can be made easier to access. Alternatively, users can learn about developers' expectations, thus learning how to use the existing system more effectively. For instance, learning that they are not expected to type full URL's in Netscape Navigator™ can lead users to omit characters such as "`http://`".

Expectation Agents

We propose an approach to remote usability data collection in which expectations are encoded in the form of software agents, called expectation agents, that monitor usage and perform various actions when encapsulated expectations are violated. Figure 1 depicts a software development process in which developers: identify usage expectations to be checked as applications are developed, create agents to monitor user interactions, deploy agents to run on users' computers, and receive feedback from agents regarding application usage, and mismatches in expected versus actual use. When mismatches are detected, agents can perform a number of actions including notifying users and developers of the mismatch, reporting contextual information as well as the user interface events leading up to and following "critical incidents", providing guidance or suggestions to users by explaining developers' expectations, and collecting feedback directly from users regarding mismatches.

Usage Scenario

Our prototype expectation-driven event monitoring system (EDEM) provides developers with tools for authoring agents, dynamic displays for visualizing the components and events of the interface being monitored as well as agent activity, and an agent runtime system that allows agents to be downloaded to monitor user interactions on user computers, while reporting data back to developer computers.

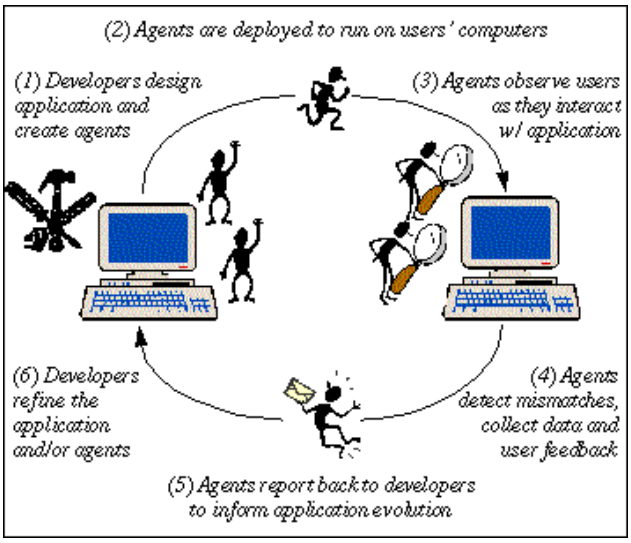

Figure 1. A development process augmented with agents for collecting usability data.

To see how EDEM might be used to collect valuable feedback from users, consider the following scenario.[1] A group of engineers are tasked with designing a web-based user interface to allow users access to a large store of transportation-related information. The interface in this example is modeled after an existing interface (originally written in HTML and JavaScript) that allows users to request information regarding Department of Defense cargo in transit between points of embarkation and debarkation. For example, an officer might use the interface to determine the current location of munitions that he/she ordered for his/her troops in Bosnia. This is an example of an interface that might be used repeatedly by a number of users in completing their work. It is important that interfaces supporting frequently performed tasks (such as steps in a business process or workflow) are well-suited to users' tasks, and that users are aware of how to most efficiently use them, since inefficiencies and mistakes can add up over time

After involving users in design, constructing use cases, performing task analyses, doing cognitive walkthroughs, and employing other user-centered design techniques, a prototype implementation of the form is ready for deployment. Figure 2 shows the prototype interface. The designers in this scenario were particularly interested in verifying the expectation that users would not frequently change their "mode of travel" selection in the first section of the form (e.g. "Air", "Ocean", etc.) after having made subsequent selections, since the "mode of travel" selection affects the choices that are available in subsequent sections. Operating under the expectation that this would not be a common source of problems, the designers made the

1.This scenario is adapted from a demonstration performed by Lockheed Martin C2 Integration Systems within the context of a large-scale, governmental transportation information system based on the Global Transportation Network (GTN). The GTN is a system that gathers, integrates, and distributes transportation-related information and acts as the central clearinghouse of transportation information for the U.S. Department of Defense.

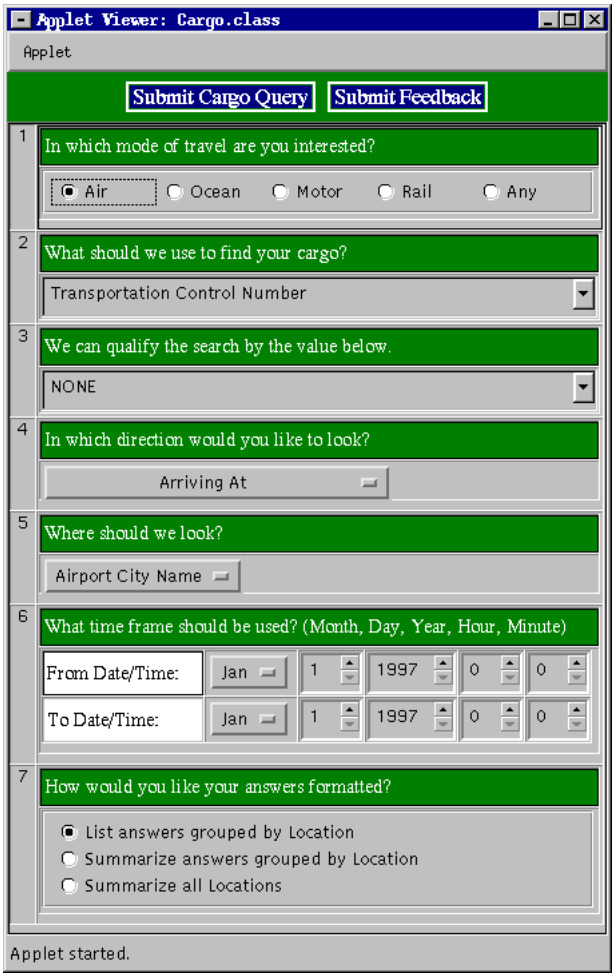

Figure 2. A prototype cargo query interface.

decision to simply reset all selections to their default values whenever the "mode of travel" selection is reselected.

The designers were then able to use EDEM facilities to create an agent that would detect whenever the user changed the mode of travel selection after having made subsequent selections. The agent was then downloaded (along with other agents) to users' computers — automatically upon application start-up — where it monitored user interactions and reported data back to developers when expectations were violated by actual usage (See [4] for more details).

When agents detect potential problems, they can report feedback to developers unobtrusively via e-mail, and may also be used to initiate direct communication between users and developers. In this case, the designers configured the agent to indicate to users that it had detected a violation. Users were then given the option to request more information describing why the agent had fired and to respond via e-mail with feedback if they desired. See Figure 3. The same facilities may also be invoked directly by users to volunteer information regarding problems not detected by agents. Direct and indirect user feedback was emailed to a help desk where it was reviewed by support engineers and entered into a change request tracking system. With the help

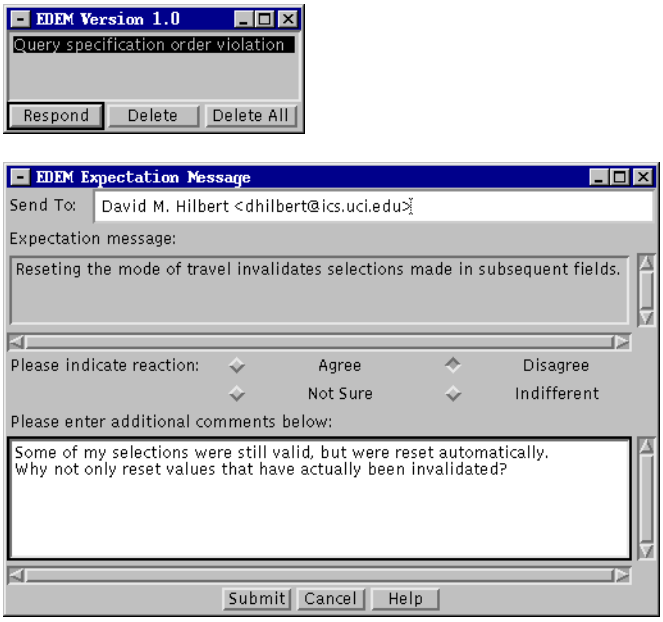

Figure 3. An expectation agent message dialog.

of other systems, the engineers were able to assist the help desk in providing a new release of the interface to the user community based on the usage information collected from the field.

It is tempting to think that this example has a clear design flaw that, if corrected, would clearly obviate the need for an expectation agent. Namely, one might argue, the application should automatically detect which selections must be reselected and direct the user to reselect only those values. To illustrate how this objection misses the mark, let us assume that one of the users actually responds to the agent with exactly this (direct) feedback. After reviewing the agent-collected feedback, the engineers consider the suggestion, but unsure of whether to implement it (due to its impact on the design, implementation, and test plans), decide to review the log of (indirect) feedback regarding expectation violations. The log, which documents over a month of use with over 100 users, indicates that this problem has only occurred 5 times, and always with the same user. As a result, the developers decide to put the change request on hold.

The ability to base design decisions on empirical data in this way, and to evaluate the importance of user feedback with respect to its likely impact on the user population at large, are key contributions of this approach. Another important contribution is the explicit treatment of usage expectations in the development process. Treating usage expectations explicitly helps developers think more clearly about the implications of design decisions. Because expectations can be expressed in terms of user interactions, they can be monitored automatically, thereby allowing information to be gathered on a potentially large scale. Finally, expectations provide a principled way of focusing feedback collection so that information is collected surrounding "critical incidents" and direct feedback can be evaluated based on how often the events leading up to them have occurred in practice.

Note that when direct communication between users and developers is initiated, it may take several forms. It may be synchronous or asynchronous, via voice, video, or electronic mail. The appropriate communication policy will depend on the development situation. For instance, synchronous groupware technologies, e.g., Internet-based video conferencing, might work well in small-scale, in-house development situations, while asynchronous communication policies might be preferable in larger-scale product development situations. When users greatly outnumber developers, information gathered from agents will need to be filtered through information management mechanisms before being presented to developers. Finally, mediator roles [2] may need to be established to manage communication between users and developers.

CONCLUSIONS

The main contributions of this work include an expectation-driven approach to feedback collection and an agent-based architecture that together make large-scale collection of user feedback over the Internet a practical possibility. Initial experience with the Global Transportation Network demonstration project indicates that valuable feedback can be captured with only modest investment on the part of developers.

By treating usage expectations explicitly in the development process, we provide a principled way of focusing feedback collection. By encapsulating data collection code within expectation agents, we allow data collection to evolve flexibly without impacting the deployment of the applications being monitored. Finally, by allowing expectation agents to perform event abstraction (described in more detail in [5]), we allow data to be filtered in a scalable way, reducing network bandwidth requirements, and allowing data collection to address events at multiple levels of abstraction. This approach has the potential of improving user-developer communication, increasing user involvement in development, and allowing developers to capture useful feedback in a scalable way. All of this could potentially be performed on a large and ongoing basis over the Internet.

ACKNOWLEDGMENTS

The authors would like to thank J. Robbins, A. Girgensohn, F. Shipman, A. Lee, and A. Turner who worked on precursors to this work and continue to provide insight and support. This work is financially supported by the National Science Foundation, grant number CCR-9624846.

REFERENCES

1. A. Girgensohn, D.F. Redmiles, and F.M. Shipman III. Agent-Based Support for Communication between Developers and Users in Software Design. In *Proceedings of the Knowledge-Based Software Engineering Conference 1994.*
2. J. Grudin. Interactive Systems: Bridging the Gaps between Developers and Users. *IEEE Computer.* April, 1991.
3. H.R. Hartson, J.C. Castillo, J. Kelso, and W.C. Neale. Remote Evaluation: The Network as an Extension of the Usability Laboratory. In *Proceedings of CHI'96,* 1996.
4. D.M. Hilbert and D.F. Redmiles. An Approach to Large-Scale Collection of Application Usage Data Over the Internet. In *Proceedings of the 20th International Conference on Software Engineering 1998.*
5. D.M. Hilbert and D.F. Redmiles. Agents for Collecting Application Usage Data Over the Internet. In *Proceedings of the Second International Conference on Autonomous Agents 1998.*
6. E.D. Smilowitz, M. J. Darnell, A.E. Benson. Are we overlooking some usability testing methods? A comparison of lab, beta, and forum tests. *Behaviour and Information Technology, Usability Laboratories Special Issue* (Ed.) J. Nielsen, Vol.13, No.1 & 2, 1994.