Me Hates This: Exploring Different Levels of User Feedback for (Usability) Bug Reporting

Florian Heller
RWTH Aachen University
52056 Aachen, Germany
flo@cs.rwth-aachen.de

Thorsten Karrer
RWTH Aachen University
52056 Aachen, Germany
karrer@cs.rwth-aachen.de

Leonhard Lichtschlag
RWTH Aachen University
52056 Aachen, Germany
lichtschlag@cs.rwth-aachen.de

Jan Borchers
RWTH Aachen University
52056 Aachen, Germany
borchers@cs.rwth-aachen.de

Moritz Wittenhagen
RWTH Aachen University
52056 Aachen, Germany
wittenhagen@cs.rwth-aachen.de

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
ACM 978-1-4503-0268-5/11/05.

Abstract
User feedback for deployed software systems ranges from simple one-bit-feedback to full-blown bug reports. While detailed bug reports are very helpful for the developers to track down problems, the expertise and commitment required from the user is high. We analyzed existing user report systems and propose a flexible and independent hard- and software architecture to collect user feedback. We report our results from a preliminary two-week user study testing the system in the field and discuss challenges and solutions for the collection of multiple levels of user feedback through different modalities.

Keywords
Usability, evaluation, user reports, feedback, modality.

ACM Classification Keywords
H.5.2. Information interfaces and presentation: User interfaces – Evaluation/methodology

General Terms
Experimentation, Human Factors

Figure 1. The facebook like-button allows to give very basic feedback.

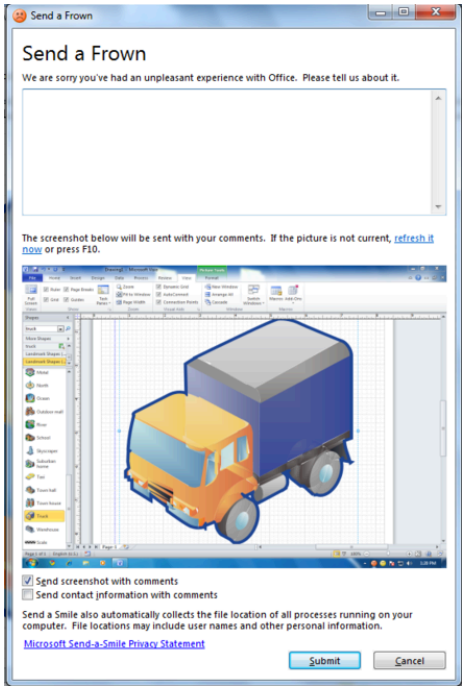

Figure 2. The Send-A-Frown application from the MS Office 2010 beta. The user can send comments with an optional screenshot to the developer team

Introduction

Software usability and user centered design have been at the center of our research community's efforts for many years. Despite considerable progress, however, non-optimally designed user interfaces and frustrated users are still a common sight. Many software bugs and usability problems are only found when the software has already been deployed and is in active use. Thus, developers have to rely on user feedback mechanisms to find, investigate, and fix such problems.

While this procedure is common practice, a lot of users do not file bug reports or enhancement requests when they encounter frustrating situations (crashes are an exception as these often auto-generate bug reports). Reasons for this may include the lack of incentive to report usability problems, lack of trust that the feedback will have any impact, feedback mechanisms that are unknown to the user or that are difficult to find, and general unwillingness of frustrated users to manually fill out bug report forms. Instead, users tend to blame themselves or learn to accept cumbersome workarounds to avoid these problems [10].

To include user feedback at these late stages of a software product's life cycle, user feedback mechanisms should be easy and quick to invoke, be presented in a consistent fashion among different products, and impose a minimum amount of inconvenience to the user who is probably in a negative emotional state. We present a number of experimental prototypes of such feedback mechanisms which allow users to express their disapproval of the current state of their system in different ways.

Collecting User Feedback

Current techniques for collecting user feedback in deployed software systems cover a wide spectrum, ranging from classic bug reporting facilities which are often external tools to the minimalistic but ubiquitous content-related feedback mechanism of today's social networks.

Traditional bug reports are usually targeting software developers rather than end users. Bug reporters are usually expected to provide information such as detailed problem descriptions, source code examples, or bug severity levels. The average user, however, in many cases may not be able or willing to provide this level of detail. In addition, bug reporting facilities are often not integrated with the users' workspace but require launching special tools or websites. This results in only a small part of users participating in reporting problems [9].

In contrast, simple *one-bit-feedback* facilities such as the facebook "like" button (**Figure 1**) or the YouTube thumbs-up/thumbs-down mechanism are used millionfold; they are easily visible, have a rather consistent representation, and only require a minimum amount of effort on the users' side. Of course, these mechanisms also do not provide much information—they only link a sentiment to the context the feedback button refers to.

Examples in between these extremes exist. Recently, the Microsoft Office 2010 beta (**Figure 2**) and the Firefox 4 beta (**Figure 3**) have included feedback UI elements that are always visible when the user is working with the software. Clicking these elements users are presented with a dialogue that asks to give a

short statement about what they liked or disliked about the software in that moment. It is important to note that most of the context, e.g., the current state of the application or an optional screenshot, is captured automatically and included in the report. This allows meaningful reports to be generated by non-expert users while reducing user effort at the same time.

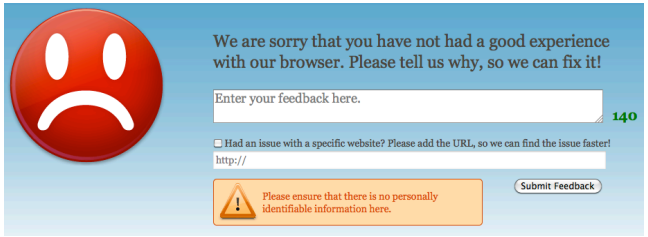

Figure 3. Feedback website for the Firefox 4 beta. Instead of relying on detailed information the feedback is limited to 140 characters and a URL.

Research Questions

Intuitively we can see that there is a tradeoff between two forces across the described spectrum of user feedback reporting techniques. On the one hand, bug reports provide software engineers and UI designers with in-depth information on the context and effects of the problems in their products but are demanding time, expertise, and commitment from the reporting users. One-bit-feedback, on the other hand, imposes no such requirements on the users but will leave the developers and designers only with an 'opinion histogram' or 'problem heatmap' on the different states of the software.

Two central questions arise from this tradeoff that have to be answered before an informed decision about how end user feedback should be included in deployed software can be made:

a) How does the amount of effort that is expected from the user influence the number of submitted reports?
b) Can a larger number of user reports that contain less user-generated information (but may contain automatically gathered data) be as valuable for developers and designers to find and fix problems in deployed software?

Another point that we have to consider is that when users encounter problems while working with computers they are likely frustrated and in an emotional state of stress. This suggests that it may be beneficial to present a reporting UI not as part of the software that caused the problem but as part of a trusted entity, e.g., the operating system or even an external device. The latter could even offer the possibility for emotional relief by venting if the device was built in a sturdy fashion and could endure physical punishment. We add this as a third and fourth question to the list:

c) Does the modality of the feedback mechanism (software button vs. hardware device) influence the number of submitted reports?
d) Does a hardware artifact that can be punched offer emotional relief to frustrated users?

Related Work

Previous work [3], [2], [8], [7] researched usability evaluation of deployed software. They found that end users can provide information that is adequate for the developer to identify the usability bugs [3], [2], [1].

Figure 4. The hardware button

However, reporters desire simple and short reports [3], [2], [8], [7] because "users are typically more concerned with getting their work done than in paying the price of problem reporting while developers receive most of the benefit. As a result, often only the most obvious or unrecoverable errors are reported." [4], [9]

With increasing number of bug reports it becomes necessary to preprocess them and to detect duplicates. Ko et al. [6] gathered a large corpus of bug report titles and found that these short headings are already expressive enough that automated tools can identify the component and concern in question. The Firefox beta (**Figure 3**) even allows reporters only to report short statements.
A challenge for automatic recordings is that the time between the bug incident and the filing of its report is often too large to allow collection processes to capture the problem [3]. Also, most errors are usability based and collected data is often inadequate to identify user experience problems [7].

Whether venting has a lasting impact on the emotional state of the angered reporter is debated [5] but might also increase satisfaction and perceived product value [11]. Nichols [9] suggests that developers let the reporter know that (and how) her report impacted the development to encourage their involvement.

Prototype Design

We built a first prototype feedback system for Mac OS X. Our prototype consists of a background process collecting system information and an optional hardware button (**Figure 4**). In case the hardware button is not connected, an on screen button is displayed in the OS X menu bar (**Figure 5**). We decided on this position over a floating window or a separate program to ensure constant accessibility of the button. This reduces the likelihood of the button itself adding to the user's frustration.

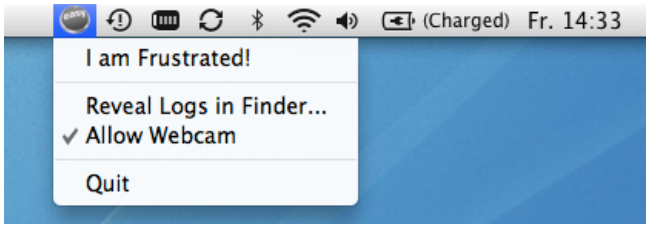

Figure 5. The on-screen button of our prototype

The software is a background process that monitors certain system characteristics and packs them into an incident report whenever the user clicks on the on-screen button or punches the hardware button. For this first prototype we did not collect additional user information after the button is pressed. The information contained in such a report consists of a (full-screen) screenshot, a list of currently running applications, system load information, current mouse position, the optional user comment dependent on the mode of the tool, and, if allowed by the user, a webcam picture. The report is stored locally on the user's system.

Two modes are available: one that opens a text view after the user has invoked the tool to query her for additional information (much like the Firefox beta does, cf. **Figure 3**) and one where no extra action by the user is required. In the latter case, users were provided with a management tool for the generated incident reports, which allows them to select any number of prepared reports and modify, annotate, delete, or send them off for evaluation. Our prototype implementation

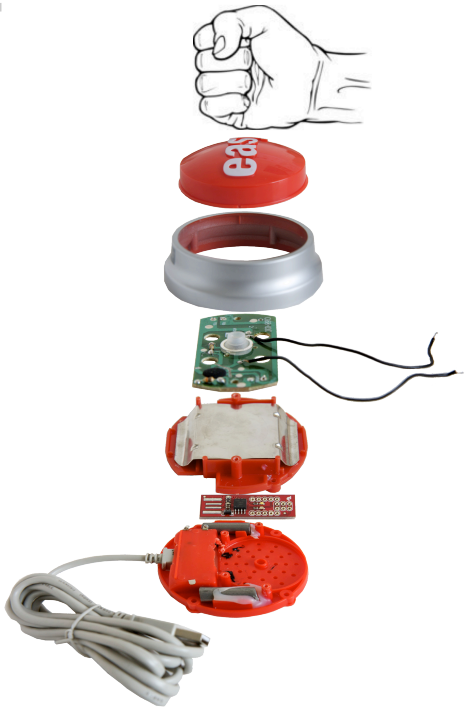

Figure 6. Detailed view on the components of the hardware button. The AVR microcontroller on the small PCB registers at the system as USB HID and reports the state of the switch on the green PCB.

is only available for Mac OS X but a Windows version is under development.

The hardware (**Figure 6**) consists of a simple mechanical switch and an AVR microcontroller, which runs a software USB-stack[1] and registers at the system as a human interface class device. Our background process searches for our specific hardware IDs and tracks incoming events.

Pilot Study

We conducted a two-week preliminary study with 10 computer science students. All participants used our software prototype with the button for one week and without for the other week, and were instructed to comment and send off the incident reports at the end of each day. Along with each incident report, participants had to fill out a self-reporting questionnaire, asking for, e.g., the importance of the task to be accomplished or the level of frustration that the incident caused. We analyzed the comments and ratings of the collected 65 reports. Since this was only a preliminary evaluation among a non-diverse user group, results are not generalizable, but give us an indication about possible trends that we want to study in more detail during future evaluations. Overall we see that the number of reports generated with the hardware button (41) is much higher than with the menu (24). The button was also reported as much more useful to express the encountered feeling of frustration while generating an incident report. However, only for half of the reports in the hardware button condition and only a third of the reports in the menu condition the participants reported feeling better after they pressed the button. This can be explained by the fact that, even though one can express the feeling of frustration, the problem for which the report was generated still persists.

We asked the participants to rate the importance of the task they were trying to accomplish when they pushed the button, as well as an estimate on how frustrating the encountered problem was perceived. We received more reports on incidents with a lower frustration level, with the physical button, suggesting that the hardware lowers the barrier for generating such reports.

Generally, we received very positive feedback on the physicality of the button although its usage is somewhat restricted for laptop users in the sense that they will not carry the button with their laptop all the time and thus can use the button only at a specific location.

Open research questions

With the upcoming implementation of our software prototype for Microsoft Windows the number of potential users will grow by a considerable amount.

We will conduct a series of studies to find answers to the research questions defined earlier. At first, we want to study how the amount of detail and expertise that is required to submit a report affect the number of user reports. If with a one-bit-feedback facility users report more incidents but with less detail, does the amount of reports compensate for their simplicity compared to a full user report with custom text. The goal is to find the right balance between simplicity for the user or incident

[1] http://www.obdev.at/products/vusb/index-de.html

reporter and the amount of required information for the developer to be able to track down a problem.

The second area we want to explore is the modality of the feedback. Does a physical button as an artifact independent of the currently used software represent a more trustworthy instance to generate incident reports. Another point that arose during our preliminary study is, if the physicality and the potential physical stress relief that comes with it lead to an emotional relief. We also have indications that the physicality of the button affects the number of generated reports, especially that the number of less critical reports increases. Finally, does the number of button presses allow an estimation of the severity of a problem? After submitting a full bug report for a specific problem, we usually assume that someone will take care of it and thus, we do not report the problem a second time. But does a (physical) button get pressed every time a specific incident occurs?

Acknowledgements

This work was funded in part by the German B-IT Foundation and by the German Government through its UMIC Excellence Cluster for Ultra-High Speed Mobile Information and Communication at RWTH Aachen University.

References

[1] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann. What makes a good bug report? In *Proc. ACM SIGSOFT* '08, ACM Press, 308–318, 2008.

[2] J. C. Castillo, H. R. Hartson, and D. Hix. Remote usability evaluation: can users report their own critical incidents? In *Proc.* CHI '98, ACM Press, 253–254, 1998.

[3] H. R. Hartson and J. C. Castillo. Remote evaluation for post-deployment usability improvement. In *Proc. AVI '98*, ACM Press, 22–29, 1998.

[4] D. M. Hilbert and D. F. Redmiles. Large-scale collection of usage data to inform design. In *Proc. INTERACT'01*, IOS Press, 569–576, 2001.

[5] J. Klein, Y. Moon, and R. Picard. This computer responds to user frustration: Theory, design, and results. *Interacting with computers*, 14(2): 119–140, 2002.

[6] A. J. Ko, B. A. Myers, and D. H. Chau. A linguistic analysis of how people describe software problems. In *Proceedings of the Visual Languages and Human-Centric Computing*, IEEE Computer Society, 127–134, 2006.

[7] B. Murphy. Automating software failure reporting. *Queue*, 2:42–48, November 2004.

[8] D. Nichols, D. McKay, and M. Twidale. Participatory Usability: supporting proactive users. In *Proc. CHINZ'03*, ACM Press, 63–68, 2003.

[9] D. Nichols and M. Twidale. The Usability of Open Source Software. *First Monday*, 8(1), 2003.

[10] D. A. Norman. *The design of everyday things*, Basic Books New York, 2002.

[11] P. Nyer. An investigation into whether complaining can cause increased consumer satisfaction. *Journal of Consumer Marketing*, 17(1):9–19, 2000.