Ludwig-Maximilians-Universität München
Department "Institut für Informatik"
Lehr- und Forschungseinheit Medieninformatik
Prof. Dr. Heinrich Hußmann

Diploma Thesis

System Malfunction Recognition and Response Based on User Behavior

Tobias Hößl
tobias@hoessl.eu

Time frame:	01.11.2007 to 30.04.2008
Supervisor:	Dipl.-Medieninf. Raphael Wimmer
Responsible professor:	Prof. Dr. Heinrich Hußmann

Abstract

Computer systems that observe user behavior in order to detect malfunctions and respond to them are mostly unexplored from a scientific point of view until now. However, they may hold much potential to improve human-computer interaction. This thesis proposes a reference model for such systems. It describes common computer problems and their symptoms, reactions by the users and ways to respond to such problems on system side. Taxonomies covering common use-cases are provided and both psychological and technical aspects are covered.
In order to show the effectiveness of behavior-based malfunction response, three systems were implemented: the Microphone Helper assists the user in configuring the microphone. The Mouse Helper recognizes confused user behavior in situations of mouse cursor freezes. The "Frustration 2.0" project tries to prevent user frustration when encountering buggy JavaScript-based websites and provides the manufacturer of the website with diagnostic information. The first two projects were evaluated empirically in user studies: the detection rate and the time gains were measured and the opinions of the users about such systems were analyzed. The results indicate that behavior-based malfunction response can be helpful in many cases, but they also show specific problems that have to be considered when designing such systems.

Zusammenfassung

Computer-Systeme, die das Benutzerverhalten beobachten, um eigenes Fehlverhalten festzustellen und darauf zu reagieren, sind bislang wissenschaftlich weitgehend unerforscht, obwohl sie die Mensch-Maschine-Interaktion möglicherweise deutlich verbessern könnten. Um weitere Forschung anzustoßen, schlägt diese Arbeit daher ein Referenzmodell für solche Systeme vor und beschreibt häufig auftretende Fehler in Computern und deren Symptome, häufige Benutzerreaktionen und Möglichkeiten, seitens des Systems darauf zu reagieren. Klassifikationen für verschiedene Anwendungsfälle werden vorgestellt und sowohl psychologische als auch technische Aspekte des Themas werden beleuchtet.
Um die Effektivität von verhaltensbasierter Fehlerbehandlung zu zeigen, wurden drei Systeme implementiert: der "Microphone Helper" unterstützt den Benutzer bei der Konfiguration des Mikrofons, der "Mouse Helper" erkennt im Falle eines hängenden Mauscursors typische Verhaltensweisen des Nutzers, und das Projekt "Frustration 2.0" versucht bei fehlerhaften JavaScript-basierten Webseiten eventuelle Frustration des Nutzers zu verhindern und den Hersteller der Seite mit Problemberichten zu unterstützen. Die ersten beiden Projekte wurden empirisch durch Benutzerstudien evaluiert: die Erkennungsrate und die gewonnene Zeit wurden gemessen und die Benutzer nach ihren Meinungen zu solchen Systemen befragt. Die Ergebnisse weisen darauf hin, dass verhaltensbasierte Fehlerbehandlung in vielen Fällen hilfreich sein kann, es aber auch spezielle Probleme gibt, die beim Design solcher Systeme bedacht werden müssen.

Contents

1 Introduction

1.1 Computers and errors

In the last couple of years, both stability and usability of computer systems have come a long way. Whereas the infamous "Blue Screen of Death" is still well-known as a symbol for computer crashes, the results of a pre-study for this thesis suggest that the real thing is hardly seen anymore. The disillusioning fact is, however, that even with less of those fatal errors, working with computers still can be annoying and frustrating, as many non-fatal malfunctions keep appearing.

While reducing the amount of errors in computer programs, making the hardware more reliable, and paying more attention on usability certainly is the most important way of dealing with problems beforehand, it is unlikely that a "perfect" computer system is going to come along in the foreseeable future, and we will have to live with a certain amount of malfunctions for the time being. Several arguments are supporting this assumption:

- Computer systems are designed by humans and humans just do make mistakes.
- No hardware being used for computer systems is perfectly reliable.
- Some technologies have inherent limitations. As a very obvious example, all battery-powered devices will stop working if they are not recharged.[1]
- Some systems have intended restrictions or purposely confusing behavior set upon them. For example, until recent events[2], it was a deliberate decision not to have mobile phones reception in Munich's subways, due to concerns about disturbances and radiation[3]. Nevertheless, from the perspective of a user who is talking on the phone while walking into the station, the forced termination of the call when losing the signal can be seen as a malfunction of the system.
- For some situations, the only distinction between "working perfectly" and "working in an unintended way" is the expectation of the user towards the system. If the laptop screen remains blank after switching it on (reason: it is switched to "TV-Out only"), this can be exactly what the user wants - but it can also be really irritating for the user, making it a "perceived malfunctioning".

1.2 Behavior-based malfunction response

So not only will malfunctions continue to happen; sometimes it is also difficult to impossible for a computer system to decide on its own, if its current behavior is being perceived as malfunctioning or not. If a computer system is supposed to react on such malfunctions and help the user, while avoiding "false alarms", it is helpful to know what the users thinks about the current situation. While it is, from a technical and ethical point of view, mostly impossible to observe the user's thoughts, it is possible to watch certain artifacts of his behavior and draw conclusions out of it.

The basic question of this thesis therefore is: as malfunctions will always occur and can not always be recognized as such by the system alone, how can knowledge about the user's behavior help to decide if the system is malfunctioning and how to properly react to it?

There are many possible applications of taking the user's behavior for malfunction recognition and response. In this thesis, three applications are covered in detail:

- The user wants to use the microphone of his computer. However, it is muted for recording by an uncommon combination of mixer settings. While trying to figure out the problem, the user occasionally speaks, blows into or taps onto the microphone. The computer notices that the user wants to use the microphone, and presents the correct setting to him.

[1]It might be up to discussion if an example that obvious can be seen as an actual malfunction. This thesis, however, takes a rather broad view on the concept of "malfunction".
[2]`http://www.sueddeutsche.de/muenchen/artikel/499/152115/`
[3]`http://www.teltarif.de/arch/2007/kw28/s26561.html?page=2`

- A stylish “Web 2.0”-site publishes a new version of a JavaScript-based drop-down-menu, which is incompatible to the user’s current browser. The user tries to open it by moving the mouse cursor over the menu element and clicking on it, however nothing happens. Because the browser cannot fix this problem, an information window appears, apologizing to the user and providing the webmaster of this site with details about the occurred error.

- The mouse cursor freezes at one position or is jumping between several positions on the screen. As a common response, the user rapidly moves around the mouse, like shaking it. As this behavior is typical for error situations, the computer notices the malfunction and reacts by reinitializing the mouse driver.

These applications serve as an example for the benefit of behavior-based malfunction response and are evaluated more in detail in this thesis. Whereas the benefits of such systems are mostly unexplored yet, a couple of applications which could be seen in the light of this already do exist besides the three mentioned before. To name two of them:

- When a program is frozen in the desktop system KDE and the user tries to quit the program using the window manager KWin’s commands, the system notices that the program can not shut down normally anymore and therefore asks the user if it should kill the program forcefully.

- A much more elaborate application is described by the US-patent 20070300174 by Microsoft (“Monitoring group activities” [16]): in order to detect if the user currently is having problems with the system, it monitors the task the user is currently working on, the current performance and at least one physiological characteristic of the user (like heart rate, galvanic skin response, brain signals or respiration rate).
  The system automatically classifies the kind of assistance the user needs, finds another user in the same work group who might have more experience with this specific task and is currently available, and establishes communication between these two users. Monitoring users to such an extent obviously raises question regarding privacy. To my knowledge, the described technique is not included in any finished product yet.

1.3 Terminology

Malfunction In the literature about computer problems, many terms are used more or less synonymously to describe errors: “malfunction”, “error”, “problem”, “mismatch”, “failure”, and so on. While there are subtle differences in the meaning of these terms, it is hard to find distinct definitions. In “Errors in Working with Office Computers” [31], Zapf et al. mentioned this problem and explained why they chose the term “error” over “problem” and “mismatch” - however they had a strong focus on errors made by the user. The focus of this thesis lies more on problems originating from the computer, therefore another term was preferable. An earlier version of this thesis used the term “failure”. However, when it became clear, that many computer problems this thesis deals with are rather small compared to total system failures, it was decided to use the more general term “malfunction”, in the sense of “[to] fail to work normally”[4].

As it depends on the user’s expectation what “work normally” means, having a malfunction does not necessarily mean that the computer system made a “mistake” from a technical point of view. In such cases, the term “perceived malfunction” is frequently used in this thesis to stress the dependence on the user’s expectation.

Behavior-based malfunction response A short form of “system malfunction recognition and response based on user behavior”. While some aspects (like the recognition) are not clear from this short form, it refers to the same kind of system and just tries to make the the text easier to read.

[4]http://www2.merriam-webster.com/cgi-bin/mwdictsn?va=malfunction

Rapid mouse movements This term refers to the specific kind of behavior that users make when the mouse cursor freezes (as shown in Chapter 5.7.1): making short, hasty movements with the mouse, like shaking it.

False alarm This term refers to a situation, in which the system believes it is in a malfunctioning state and triggers error recovery, although it is not.

1.4 Outline of this thesis

The main contribution of this thesis lies in proposing a theoretical framework for incorporating user behavior into the error recognition and response process. Surprisingly, to my best knowledge, no such systematization existed yet.

This research topic borders to many other research fields; related projects and articles are shortly described in Chapter 2 of this thesis.

The basic model of the provided theoretical framework is explained in Chapter 3, before covering different aspects of it more in detail: Chapter 4 first shows an error taxonomy developed by Zapf et al., then explains a qualitative survey conducted for this thesis to find the problems that are annoying computer users nowadays, and a taxonomy of computer errors based on the findings of this study.

Chapter 5 shows different kinds of user reactions on computer problems. First, a classification of behavior based on the results of the survey is presented, before strategies user follow to recover from errors are shown. Computer frustration is covered more in detail as well as aggressive behavior towards computers. Adaption and superstitious behavior is shown and the results of two small user-studies with specific applications in mind are presented.

The theoretical part is concluded by Chapter 6, which explains possibilities of system responses. They are seen from two perspectives: first an activity-based perspective, providing a taxonomy of actions that can be used for malfunction response, then a component-based perspective that can be used to decide which component in the system is responsible for the malfunction response for a given problem.

Chapter 7 explains the practical projects implemented and evaluated for this thesis. First, it is explained why they were chosen and how they are different from each other. The following three sub-chapters describe the three projects: Chapter 7.2 the Microphone Helper, Chapter 7.3 the Mouse Helper, and Chapter 7.4 the "Frustration 2.0" project.

Finally, the results of the studies accompanying two of the projects are discussed in Chapter 7.5 and an overview over possible future research is provided in Chapter 8.

2 Related Work

The most relevant fields of research for this thesis are the ones dealing with the occurrence of computer problems, with user frustration and aggressive behavior. Important works of these fields are presented in this chapter.

The research field about self-healing systems only appeared to be very relevant for this work on the first sight. At closer look, hardly any findings about self-healing systems could be transferred to behavior-based malfunction response. An exception makes the article “Self-Healing in Modern Operating Systems” [28] by Shapiro; whereas the class of problems Shapiro talks about are much more low-level than the problems this thesis deals with, the terminology of “error symptoms”, “problems” and “diagnosis” in this thesis was chosen in reference to this article.

Before behavior-based malfunction response is implemented in commercial products, it will be necessary to further analyze ethical implications of the user observation. For affective computing, Reynold and Picard analyzed this problem in “Affective Sensors, Privacy and Ethical Contracts” [24] and proposed “ethical contracts” between the user and the computer, deciding which emotions the computer is allowed to observe.

2.1 Computer problems

“User strategies in recovering from errors in man-machine systems” [12] examines the different processes of recovery that users follow when encountering an error. The errors can be either mistakes (wrong plans) or slips (wrong actions) by the user. The basic steps are error detection, error explanation and error correction. Referring to a classification by Mo and Crouzet [18], the goal of error recovery is either backward recovery, forward recovery or compensatory recovery. Five groups of error recovery strategies are identified: inner feedback, exploring system feedback, external communication, planning behaviors and error informed strategies. Chapter 5.2 of this thesis covers this framework more in detail.

“Error Recovery Representations in Interactive System Development” [10] proposes another model of error recovery, with the dimensions “error additional cost”, “system state degradation” and, optionally, “time”.

“Designing for error” [15] discusses how systems should react to errors induced by the user, taking some Unix commands as (bad) examples. The main point of this article is that the computer should not blame the user for erroneous input but itself for being unclear to understand, and assist the user to find the solution.

In “Errors in working with office computers” [31], Zapf et al. created and validated a general taxonomy of errors that occur when working with computers: functional mismatches, usability mismatches, inefficiencies and interaction errors (referring to the interaction between humans). Functional mismatches is the class most relevant to this thesis and can be further divided by the result for the user: action repetition, interruption, detour and blockade. The taxonomy is covered in Chapter 4.1 of this thesis. “Applying a Taxonomy of Error in Usability Testing” [8] by Hyland and Vrazalic shows that this taxonomy provides a helpful framework for usability testing, using a web application as an example.

Several studies have been conducted about the spread and frequency of computer error occurrence. Two studies involving Lazar, Bessiere and Shneiderman figured out by empirical research what applications created how many frustrating experiences and how much time was wasted for error recovery. In “Determining Causes and Severity of End-User Frustration” [6], they found that the applications causing the most frustration were web browsing, e-mail and word processing and the time lost because of such experiences was around 38%. In the follow-up research paper “User Frustration with Technology in the Workplace” [13], the authors elaborate more about the effects of frustrating experiences and how well users know how to handle them. “Social and Psychological Influences on Computer User Frustration” [3] explains the aspects of computer frustration from a more psychological perspective.

2.2 Aggression, frustration, emotions

Aggressive behavior against computers, one of the possible outcomes of frustration, got attention by several researchers. One of the most-cited studies is “Rage Against the Machine”, conducted 1999 in Britain in behalf of Compaq [19] [5]. It states that verbal and physical abuse of computers is a common phenomena.

One of the most exhaustive works in the field of “Computer Rage” is the thesis “Agression gegen Computer” [4] by Brinks, outlining much of the research done on computer aggression and contributing a model about factors of computer aggression. The correlation between computer frustration and aggression were evaluated using an online survey. Some of her findings are shown in Chapter 5.4 of this thesis.

The Laboratory for Automation Psychology of the University of Maryland is investigating “Computer Rage” using an online survey [6]. The results are presented on their website [21], along with illustrations about this topic that are scientifically less valuable, but entertaining nevertheless.

To reduce the frustration of the user, Picard explains in “Toward computers that recognize and respond to user emotion” [22] possibilities of computers that are emotionally aware. “This computer responds to user frustration” [11] by Klein et al. explores systems that actively support the user to manage and recover from negative emotional states. Several design guidelines and strategies for agents providing active emotion support for frustrating situations are given. Chapter 5.3.2 of this thesis shows some of their findings. A prototypical agent was implemented and positive effects on the user’s willingness to go on working with the computer were shown.

Much research is done about ways to measure the user’s current emotional state. In his master’s thesis “The Sensing and Measurement of Frustration with Computers” [23], Reynolds shows different ways to sense frustration of the user. He differentiates between active sensors that users are actively interacting with (Voodoo Doll, Thumbs Up) and passive sensors that detect the user’s mood without being used explicitly (PressureMouse, Yelling Detector). Some of these sensors are described in Chapter 5.3.2 of this thesis.

The thesis “Automatische Emotionserkennung aus sprachlicher und manueller Interaktion” [27] by Schuller categorizes emotions and goes into great detail about recognizing them by feature extraction and pattern matching of acoustic and linguistic signals, and of manual inputs like mouse cursor movements.

Mentis and Gay [17] demonstrated that pressure on a touch pad can be used to detect negative affect of the user. By “Frustrating the User On Purpose” [26], by purposely adding emulated errors into a time-critical game, Scheirer et al. measured several physiological signals of users in frustrating situations. A similar study is “Frustrating Computer Users Increases Exposure to Physical Factors” [9], showing several physical symptoms of users (pressure onto the mouse, muscle activity) when encountering frustrating situations.

[5] The original paper is not to be found online anymore, therefore a summary of MORI, the company which conducted the study, is used as secondary source.

[6] `http://lap.umd.edu/computer_rage/`

3 Basic model

As stated in the introduction, there exists no theoretical framework yet on how to incorporate user behavior into the process of malfunction recognition and response. Therefore, an initial model trying to systematize the components and possibilities of behavior-based malfunction response was developed for this thesis. It is illustrated in Figure 3.1.

The model consists of several components which are introduced briefly in this chapter, before going more into detail in the Chapters 4 to 6.

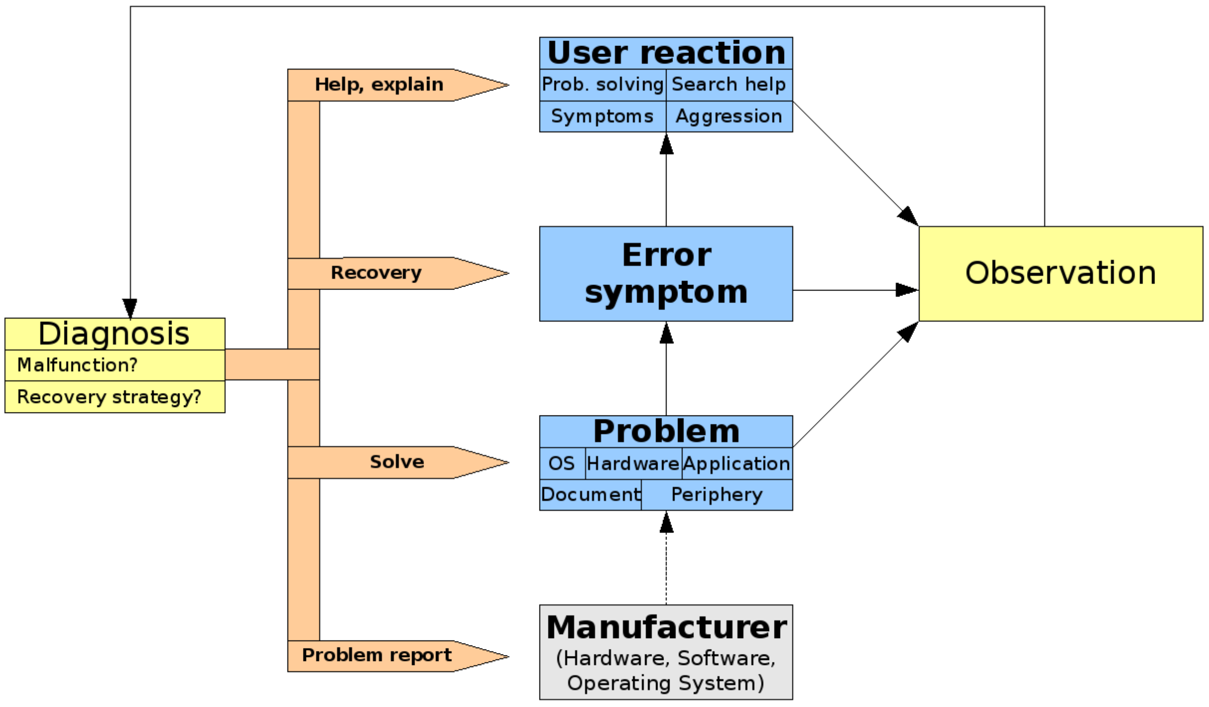


Figure 3.1: The basic model of behavior-based malfunction response, developed for this thesis.

3.1 Problems and error symptoms

Most works about computer errors mentioned in Chapter 2.1 do not differentiate between the actual problem and the symptom caused by it. As this thesis also takes a technical approach to this topic, it seems to be necessary to make a distinction between these two aspects.

The difference can be explained most easily by an example: the mouse plug accidentally slips out of the port. The user does not notice it, but sees that the cursor is not moving along the mouse's movement anymore. In that case, the plug not being connected anymore is the problem, while the cursor not moving is the symptom. Therefore, a symptom is how a problem presents itself to the user.

A problem can lead to several symptoms, and, more important, different kinds of problems can lead to the same symptom. At first glance, also driver problems, a complete system crash, or some serious hardware damage inside the mouse could lead to the same symptom as the slipped out cable at first sight.

In situations where the underlying problem is more obvious to the user, it is not as important to make this distinction.

Problems can be in different components of the whole system: The basic hardware (the set of hardware components absolutely for the computer to run), peripheral devices, the operating system, running applications, and the documents (for example a malformed documents or inconsistencies between the document and the application).

3.2 User reactions

User reactions are triggered by the error symptoms (not the problem). The reaction by the user can be roughly be divided into inner and outer reaction; the best known inner reaction is frustration, other reactions comprise confusion, adaption, withdrawal and superstition. Outer reactions cover aggressive behavior, active problem solving (or at least trying to do so), and behavioral and physical artifacts of inner reactions (physical reactions to frustration, behavior shown only due to the inner confusion).

3.3 Observation

Observation units are monitoring the current system state and the user behavior. Many problems can be observed directly by the operating system; this is already done extensively in modern operating systems (for example observing the network status, some printer malfunctions and so on). Also some problem symptoms can be observed by the system, mostly using some sort of watchdog component (periodically checking if a certain program is responding properly; failing to respond is a symptom, the underlying problem could be some programming error).

While a lot of work has been done making computing more reliable by detecting errors through means of these two observation channels, this thesis concentrates on cases in which the user behavior is an additional source of information. Observing the user can be done by various means, from analyzing keystrokes and mouse movements, over microphones and video cameras, to measuring skin conductivity and heart beating rate.

3.4 Diagnosis

Based on the data provided by the observation, a diagnosis about the current system state is made. The first and most important decision to make is if there is an malfunction at all. If nothing indicates a malfunction, the operation continues as normal. If there is clear evidence for a malfunction, for example when a problem can be observed directly by the system, the system can immediately start to respond accordingly. However, if it is not clear to the system if there is a malfunction, it has to weight all information available (notably information about the user's behavior), and either make an elaborate guess or actively ask the user for more feedback.

In case there actually is a malfunction, the system has to decide on an appropriate error handling routine. In the examples presented in this thesis, the decision how to react on a malfunction is done according to pre-defined rules. It is up to further research to decide if there is a way to deduce the appropriate system responses (semi-)automatically; a prerequisite for this would be, in my opinion, to have an elaborate formalization of the possible malfunctions and responses. The taxonomies presented in this thesis could be a first step to such a formalization, but are by far not elaborate enough yet.

3.5 Error response by the system

The possible responses by the system can be divided into four categories, characterized by the components of the overall system they are targeting at:

- If the problem is known and the system knows how to solve it, obviously it should solve it. This is especially easy if the problem is just a wrong system setting.
- If the system only knows about the error symptoms or there is no way to solve the actual problem, it can try to reduce the symptoms, do some damage control or recovery.
- Feedback to the user is a very important way of responding to malfunctions - especially in the scope of this thesis, where it is likely that the user has already noticed the malfunction, as the error detection is based on his behavior. In uncertain situations, the system can ask the user for feedback. If something in the system is changed in the course of error recovery, the user should be informed about it. The system can and should apologize to the user

for errors, and maybe give him chances to vent his frustration. If the problem can only be solved by the user himself, the system should help him doing so. If the malfunction is unavoidable, the system might explain to the user, why it is unavoidable.

- Feedback to the manufacturer of a faulty component can be a way of letting him know that a certain problem exists. In some cases, feedback by the system can be more accurate than feedback (using web forms, e-mail or telephone) by the user, as the system might know more about the actual problem, while the user mainly sees the symptoms.

4 Problems and symptoms

This chapter first explains an existing taxonomy of errors. A survey for this thesis is presented, followed by a classification of problem sources proposed by this thesis.

4.1 Problem taxonomy by Zapf

A general classification of errors that can occur when working with computer systems was created by Zapf et al. in "Errors in Working with Office Computers" [31]. The categories are illustrated in Figure 4.1.

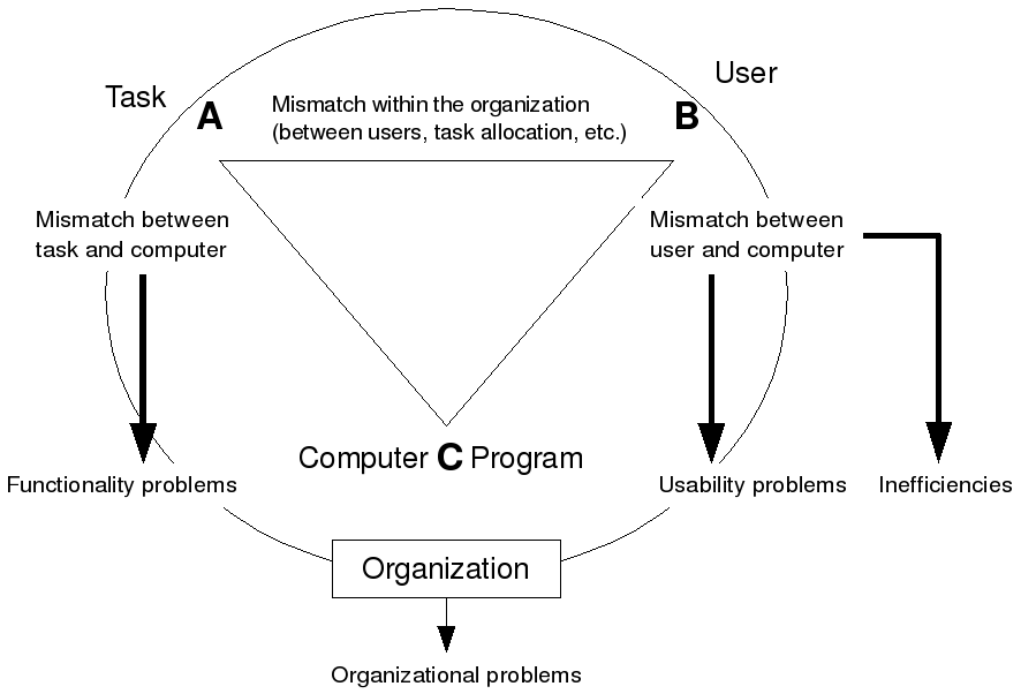

Task
A
Mismatch within the organization (between users, task allocation, etc.)
B
User
Mismatch between task and computer
Mismatch between user and computer
Computer C Program
Functionality problems
Usability problems
Inefficiencies
Organization
Organizational problems

Figure 4.1: Overview of error categories by Zapf et al. [31]

The errors are divided into four classes:

Functional mismatches

Functional mismatches occur if the system does not have the functionality the user expects it to have or if the functionality is erroneous. For the user, this can have four consequences:

- **Action repetition**. The user has to do some work again due to data loss.
- **Interruption**. The user has to perform another task before he can continue doing what he was about to do before.
- **Detour**. The user can still reach the original goal, but has to take another path than originally thought.
- **Blockade**. The user cannot finish the task he wanted to do and has to give up or change his goal.

Usability mismatches

In the context of this taxonomy, usability mismatches are errors made by the user. They are further subdivided two-dimensionally by the level of action regulation and the steps in the action process. In the axis of action regulation, the intellectual level of regulation refers to the high-level planning stage to reach a certain goal. Flexible action patterns refer to "ready-made action programs available in memory" which are deployed by the strategy developed in the intellectual level. The sensorimotor level refers to low-level, rather unconscious and automatic movements. The steps in the action process are the planning stage, the monitoring stage and the feedback stage.

The taxonomy of usability problems by Zapf et al. is shown in Table 4.1.

	Goals / Planning	**Monitoring**	**Feedback**
Intellectual level of regulation	Thought errors	Memory errors	Judgment errors
Level of flexible action patterns	Habit errors	Omission errors	Recognition errors
Sensimotor level of regulation	Sensimotor errors		

Table 4.1: Types of usability mismatches in Zapf et al.'s error taxonomy.

Inefficiencies

Inefficient behavior means that there actually is a faster way to reach the goal, but the user does not take it. This can either be due to a lack of knowledge, or out of habit.

Interaction errors

Errors that occur as a result of the communication between individuals. Unclear task allocations and a lack of coordination in the organization are given as examples.

4.2 User study

To get an initial understanding of which malfunctions are bothering users, two user studies were conducted for this thesis: a web survey and personal in-depth interviews.

Both studies used a qualitative approach, as the goal was not to get quantifiable results, but a broad overview about this topic. The in-depth interviews were chosen as an additional source of information, as it was suspected that in the low-involvement situation of a web survey, users tend to think only about the really big annoyances. The goal of the two in-depth interview was therefore to find rather small and not-so-obvious problems the users are dealing with.

Due to the qualitative approach of the web-survey, mostly open questions using multi-row text fields were used, with rather general questions. The questionnaire was divided into four parts, presented in the following order:

- Problems the user is encountering when dealing with computers. It was separately asked for problems with hardware, software, the Internet and the operating system, and asked for problems with the mobile phone.

- The user's reactions to such problems and aggressive behavior. To mask effects of social desirability response set, the participants were asked to write down behavior observed at third persons (friends and colleagues). The results of this part are presented in Chapter 5.1.

- The frequency of occurrence of some specific error situations. These questions were done in multiple-choice style, so the results can be quantified.

- A very limited amount of socio demographical information, including age and computer experience.

The complete questionnaire is presented in Appendix A.1.

The survey was online from December 6th to December 10th of 2007. Altogether, 131 users participated. Not every user answered every question. The self-reported average computer experience was 3.46, on a 5-point Likert scale from 1 ("Beginner") to 5 ("Expert"). The average age was 19.7 years, 80% of the participants were female. 92% were using Windows as their primary operating system.

The raw answers of the web survey, and the notes of the in-depth interviews are included in the attached CD of this thesis.

The main tendencies found in the web survey are:[7]

- Only a few participants were complaining about complete system crashes. Notably many user said they did not experience one at all in the last time. This tendency also was confirmed in the in-depth interviews.
- However, several participants mentioned system hangs when starting up or shutting down the operating system and when resuming after putting the computer into sleeping mode.
- Most of the hardware problems origin from peripheral devices. Problems with the mouse were mentioned most often (the cursor freezes or jumps around; the battery of a wireless mouses gets empty). Because of the amount of mouse problems reported in this study, a mouse-related system was chosen in Chapter 7.3 to evaluate the benefit of behavior-based malfunction response. Also keyboards and printers were often mentioned as the source of problems.
- Several participants reported that their computer has problem recognizing new USB-devices.
- Several problems with sudden network breakdowns were reported - mostly (but not exclusively) regarding WLAN-connections.
- A wide range of problems and annoyances were reported for the Internet; sites that cannot be found, loaded or displayed, misleading pop-up ads, plug-ins not working properly, and browser-crashes.
- Being asked about problems with application software, virtually every user reported problems about one or another software - unlike the question for problems with hardware and the operating system, where many users reported they had no problems at all. Most of the problems are complete crashes or deadlocks of the application. The software reported most often as a source for problems was the instant messaging program ICQ. Office programs were reported relatively seldom, but that might be due to the young average age of the participants.
- Not much was reported about problems with mobile phones; if something was reported at all, then mostly complete crashes of the phone, where the users had to reboot the phone.

Based on these findings, one could come the the impression that now, that operating systems gradually are becoming stable enough not to be the biggest source of computer frustration anymore, malfunctions and design errors in application software are becoming more apparent.

4.3 A taxonomy for behavior-based malfunction response

Problems and their symptoms can be classified according to the part of the whole computer system where the problem lies in.

[7]The number of participants reporting a certain problem will not be given; a quantitative survey would be more appropriate to get accurate numbers.

In “Determine Causes and Severity of End-User Frustration” [6] and “User Frustration with Technology in the Workspace” [13], the problems are categorized as follows: Internet, Applications, Operating System, Hardware and Other. Note that in the terms of this thesis, their taxonomy is rather a taxonomy of error symptoms than a taxonomy of errors. Also, some of the problems described in their taxonomy cannot be applied to this thesis (especially “typing errors”, which cannot be seen as a system malfunction anymore, no matter how user-centric the approach is).

The taxonomy used for this thesis is similar, but some changes are made to make the classification more appropriate for the following recovery-process, as it reflects the layers of the system and therefore can be used to decide which part of the system has to deal with the error. This layered system is illustrated by Figure 4.2.

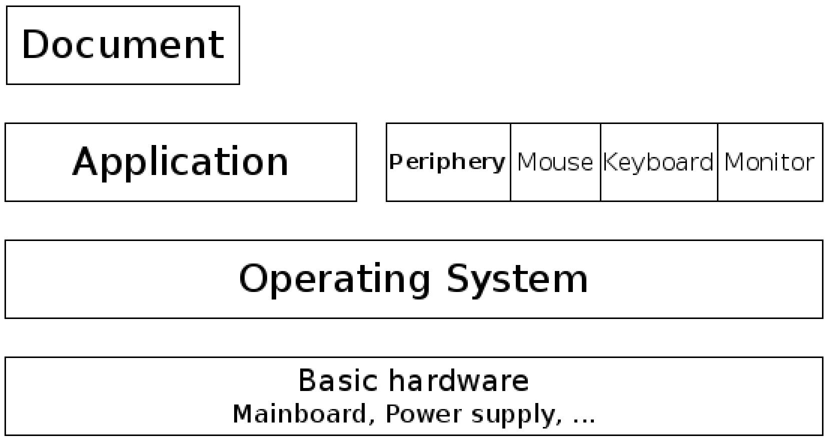


Figure 4.2: The layers in a computer system in which malfunctions can occur.

4.3.1 Basic hardware

The part of the hardware that is absolutely necessary for the computer system to work in the current configuration. The CPU and the motherboard are (usually) necessary, also the system drive, a graphics card and the power supply unit. Of course, redundant system design like RAID make single hardware components dispensable, however they do not change the conceptual reliance of a computer system on this kind of component - they just reduce the likelihood of failure.

It should be pointed out that while some additional components like physics acceleration cards surely are not necessary for a computer to work in general, once it is switched on with such a card plugged in, one usually cannot remove it without endangering the system stability, and potential hardware failures of this card can greatly affect the rest of the system. Also the famous cleaning lady stumbling over the power cable would be an error in the basic hardware.

4.3.2 Operating system

This category includes the kernel, file system, device drivers, user interface, and so on. Buggy drivers of third parties seem to be a frequent source of problems, so depending on how well the operating system can catch and handle errors produced by such drivers, they could be seen as a separate category [8].

[8] This is not done in this thesis. To analyze if it is, for example, the fault of the operating system, the driver or the actual device, if a USB-printer is not recognized properly, is beyond the scope of this thesis.

4.3.3 Periphery

This category includes all physical malfunctions that are not fatal for the whole system. Jammed printers and dropped WLAN-connections are two examples. From the system's point of view, the keyboard, mouse and monitor are peripheral devices as well. However, as those devices are usually essential for the human-computer interaction, malfunctions in these devices should be treated with special attention.

4.3.4 Application

This category includes all malfunctions (programming errors, misconfiguration, etc.) originating in the application the user is actually working with.

4.3.5 Document

Documents are information units loaded into the application the user is working with. Malfunctions can occur because of malformed or otherwise erroneous documents, or because of active contents [9].

It should be stated once more, that if a malfunction originates in a specific component, it does not necessarily mean that this component has an error in the technical sense, as a wrong system setting could be seen as an error by the user as well.

For malfunctions due to incompatibilities, it is rather hard to define which component is "at fault" - this can only be decided by the normative force of official or de-facto-standards. In the context of error recovery, it is usually the task of the lower layer (refer to Figure 4.2) to handle incompatibilities.

[9]Although, in the case of active contents, the document could also be seen as part of the application-category.

5 User reactions

Users respond to malfunctions in many different ways. While behavioral reactions, being measurable, are more important for behavior-based error response, it is also useful to know about psychological effects that malfunctions have on the user, as they have to be considered when designing the user feedback of the system. Both kinds of reactions are therefore covered in this chapter.

Based on the results of the web-survey described in Chapter 4.2, four main classes of behavior are identified in Chapter 5.1. The problem-solving aspect is covered more in detail in Chapter 5.2, according to a taxonomy by Kontogiannis. Chapter 5.3 covers computer-frustration and how to prevent it. The state of research regarding aggressive behavior towards computer system is explained in Chapter 5.4. Chapters 5.5 and 5.6 cover adaption and superstition, before Chapter 5.7 concludes this part with two small user-studies conducted for this thesis.

5.1 Results of the web-survey

The web-survey asked three questions about how users react to malfunctions:

- "How strongly do such errors affect you and how do you react?" [10]
- "What about your colleagues or friends?"
- "How do you react on such errors on the mobile phone?"

Retrospectively seen, the second question, trying to mask effects of social desirability response set, did not provide much additional information, as most participants just told that their colleagues reacted "the same way" as themselves. Only a few reported more aggressive behavior than for themselves [11].

The reactions reported in the web-survey and the in-depth interviews can be classified into four categories: problem solving, aggressive behavior, searching for help and symptoms.

5.1.1 Problem solving

Solving it - or at least trying to - is of cause the most promising way of reacting to a problem.

The most common problem solving strategies (based on this survey) seem to be restarting-operations; restarting the application, rebooting the system, switching the peripheral device off and on again. For mobile phones, some users reported an escalation hierarchy of restart: When simple turning off and on does not help, the battery is temporarily removed. If this does not help either, the SIM-card is removed for a while.

In the context of removing the battery, a user reported to blow onto the battery before inserting it again; as it is rather unlikely that this really solves the problem of the phone crashing, this would be an example of superstitious behavior, which is explained in Chapter 5.6.

Kontogiannis developed a taxonomy of user strategies to recover from errors, which is explained in Chapter 5.2.

5.1.2 Aggressive behavior

Examples mentioned in the survey are:

- Cursing (with a couple of exceptions, nearly everyone reported to curse at the computer from time to time).

[10] All three questions are translated from German.

[11] This is coherent to the web-survey about computer aggression performed by Brinks [4]; after comparing the self-reported values with the ones about the participant's colleagues, she also suspected that most participants answered honestly.

- Hitting the keyboard, the mouse or the monitor. Peripheral devices seem to be affected more frequently than the actual computer.
- Shaking a mobile phone, or throwing it.
- Gestures of vengeance against the programmers.

There are indicators that aggressive behavior towards the computer strongly depends on the social context: some participants told that they were much more careful if other people are around.

Aggressive behavior is covered more in detail in chapter 5.4.

5.1.3 Searching for help

If the users want to solve the problem but do not know yet how to (and rebooting did not help either), they try to get help by various means:

- Searching in the Internet; typing the error message or a description into a search engine.
- Asking colleagues or friends.
- Calling the hotline, or the reparation-service.
- Reading the manual[12].

5.1.4 Symptoms

Certain patterns of behavior are typical for certain error situations but can neither be seen as aggressive, nor as problem solving, as they origin in the confusion or surprise of the user. This includes:

- Rapid mouse movements when the cursor does not react anymore.
- Clicking random keys when the computer does not seem to react anymore.
- Repeating the command if the computer seems not to have "noticed" the command before.
- Rhythmical tapping with the finger.
- Exclamations of surprise (like "Huh?" or "What's now?")[13], sighing.
- The most "geeky" symptomatic behavior reported was "an increased number of typing errors": the problem reported was that the P2P downloading-program Azureus used more upload bandwidth than allowed, while the user was working remotely on a server using SSH. The upload band with of Azureus (the problem) led to a very high network latency of the SSH-connection (the error symptom), provoking many typing errors.

5.2 User strategies for problem solving

"User strategies in recovering from errors in man-machine systems" [12] by Kontogiannis provides a comprehensive framework of strategies followed by users when encountering errors and how to support them as a system designer. While this framework was designed with really safety-critical use-cases in mind (like flying an airplane or operating a nuclear power plant), it translates well into everyday computer use.

Kontogiannis focuses rather on errors made by the user. Two types of user errors are differentiated:

[12]A rather theoretical option. Actually, it was not mentioned even once by the participants of the survey.

[13]For such short exclamations, it would be exaggerated to talk of the anthropomorphisation mentioned in Chapter 5.6.1.

- **Slips** occur at the execution stage of plans; the plan the user made was correct, however he made an error when executing it.
- **Mistakes** are errors in the planning stage.

This differentiation appears to be a more coarse-grained variant of Zapf et al.'s taxonomy (see Chapter 4.3) of usability problems, with slips being somewhat similar to sensorimotor errors.

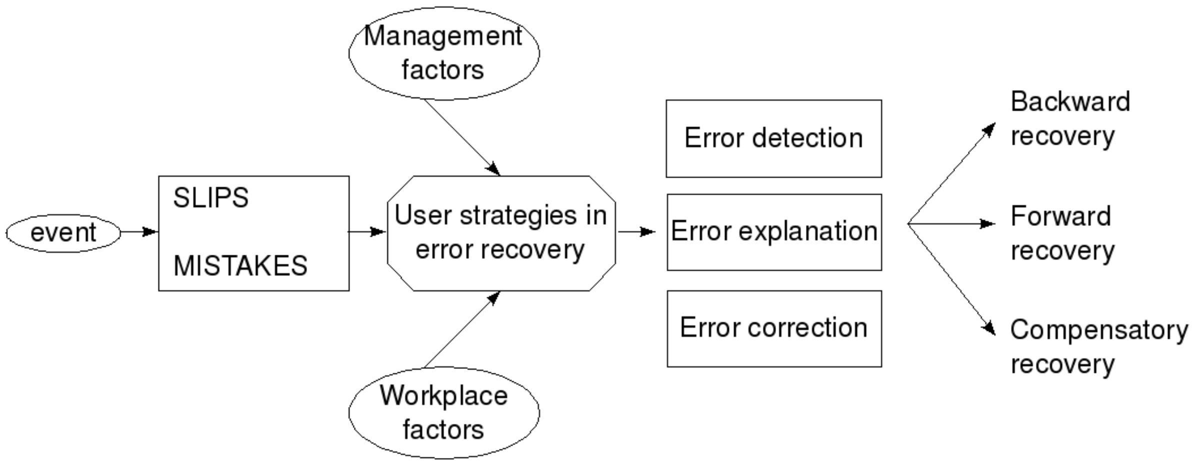


Figure 5.1: A framework for the analysis of error handling processes, by Kontogiannis [12].

In the terminology of this work, "error recovery" refers to the whole process of error detection, explanation and correction. The following five groups of recovery strategies therefore do not only influence how the user responds to errors, but also how he detects them:

- **Inner feedback**: some errors are detected either by memory alone (for example, the user forgot to do something and just remembers) or by action feedback (like by haptic feedback when pressing a wrong key), without seeing the actual outcome of the action. For error explanation and correction, users can just remember the right way of deduce it from their prior knowledge.
- **External communication**: communicating with other people can both help to detect a problem and to solve it. Kontogiannis' focus lies more on the detection of not-obvious errors in complex situations by other team members (who need to have *access* to the work of the user, pay *attention* to it and have some *expectations* and a *perspective* on it). For error explanation and correction, users can ask friends, colleagues or computer experts how to deal with the problem.
- **Exploring system feedback**: a common way for the user to detect errors is the feedback by the system, like alert boxes, error sounds, blinking red lights or the wrong outcome of the performed operation. The feedback is also commonly used for error explanation and correction.
- **Planning behaviors**: active strategies planned before; for example, regular system checks to detect errors, or emergency procedures.
- **Error informed strategies**: using experience gained in comparable situations and coping with the frustration of the situation.

Error correction is done with one of these three corrective goals:

- **Backward recovery**: the system is brought back into the state it had before the error happened.

- **Forward recovery**: the system is brought into an intermediate stable state and the original plan to reach the goal is altered.
- **Compensatory recovery**: redundant equipment is deployed, therefore the system is brought into the originally intended state.

Kontogiannis' framework for the analysis of error handling processes is shown in Figure 5.1.

5.3 Frustration

There are several slightly different definitions of the term "frustration" [13], for example "an interference with the occurrence of an instigated goal-response at its proper time in the behavior sequence" [7]. The common element of most definitions is that the user is hindered in reaching a goal.

5.3.1 Computer frustration

According to Bessiere et al. [3], the level of frustration experienced by the user after encountering a frustrating situation depends on several factors, like the importance of the goal, the severity of the interruption and the current mood of the user. The Computer Frustration Model, which is illustrated in Figure 5.2, divides these factors into two categories - Incident Factors and Individual Factors.

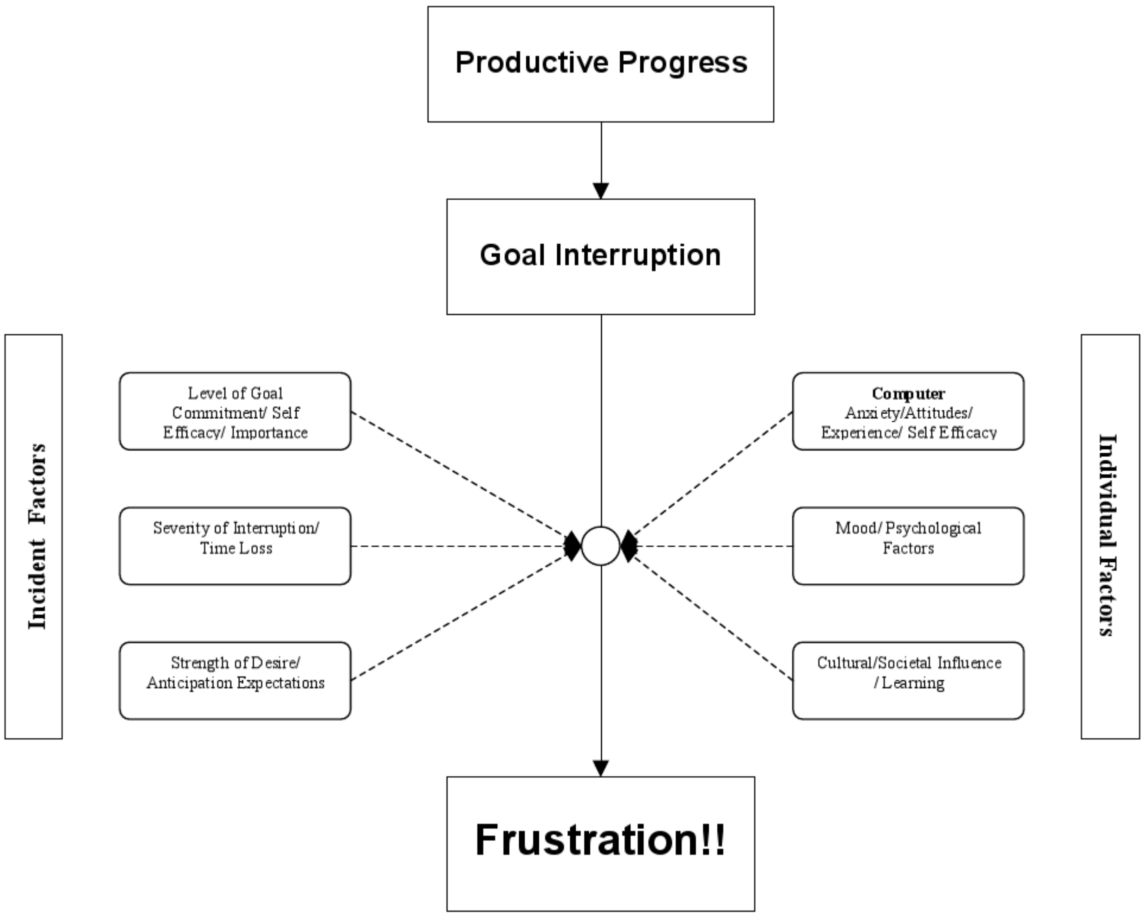

Productive Progress
Goal Interruption
Incident Factors
Level of Goal Commitment/ Self Efficacy/ Importance
Severity of Interruption/ Time Loss
Strength of Desire/ Anticipation Expectations
Computer Anxiety/Attitudes/ Experience/ Self Efficacy
Mood/ Psychological Factors
Cultural/Societal Influence /Learning
Individual Factors
Frustration!!

Figure 5.2: The Computer Frustration Model by Bessiere et al. [3]

While frustration is an psychological response at first, it can lead to physiological and behavioral changes.

Dennerlein et al. could show that "Frustrating Computers Users Increases Exposure to Physical Factors" [9]. They produced a frustrating experience in a masked user study, using an error in a long web-based survey. The participants had to fill out the survey, but right before the answers could be sent, all inputs were deleted and the participants had to start over again. Dennerlein et al. measured the muscle activity and the force applied to the side of the mouse during this test and could show, that after this frustrating experience, both were significantly increased for many users[14].

In "Frustrating the Users on Purpose" [26], Scheirer et al. showed that the skin conductivity and blood volume pressure changed at their study's participants when being frustrated. Based on these two symptoms, they built a mechanism that could distinguish between normal and frustrated state of the user correctly in about 67% of all tests.

Mentis and Gay [17] showed that users using a touch pad on a laptop pressed significantly harder on the pad after being frustrated.

According to Bessiere et al. [3], frustration can lead to different responses, which can be categorized as adaptive [15] and maladaptive. Adaptive responses refer to constructive problem solving strategies, whereas maladaptive responses are negative effects. Britt and Janus [5] categorize maladaptive responses into objective and subjective responses.

Objective maladaptive responses are:

- **Aggression** - probably the most famous response to frustration. The connection between them is discussed controversially. The Frustration-Aggression-Hypothesis by Dollard et al. [7] suggested, that frustration is the base for every aggression. However, recent studies relativised this mono causal connection; negative experiences like frustration are indeed one possible cause for aggression, but neither are they the only possible cause, nor is aggression the only effect they can lead to [20]. Barker and Dembo [1] claimed that frustration leads to regression (immature behavior) in general, and aggression is just one form of it.
- **Withdrawal**, in the sense of avoiding the source of frustration.
- **Resignation**, when the users loses all motivation to pursue the original goal.
- **Fixation**: repeating the same behavior over and over again, or getting a narrowed view on the problem. As an extreme example, Woods [30] found [16] in an analysis of simulated nuclear emergencies, "that the most common cause of failure to detect errors was fixation on the part of operators".

Subjective maladaptive responses (according to Rosenzweig [25]) are:

- **Extrapunitive response**: the user blames someone else for the occurrence of the malfunction.
- **Intropunitive response**: the user blames himself for it, feeling guilty.
- **Impunitive response**: the user does not blame anyone, but trivializes the problem.

5.3.2 Communicating frustration

Computers can not only be the source of frustration but also help the user with his emotion regulation in order to reduce the level of frustration. Computers can provide active and passive support for this [11]: for passive support, they are only used as a tool to regulate the mood - for example by surfing on YouTube or playing an action game after a frustrating experience to relieve stress. In this case, the computer does not become active on its own and the negative feelings are not directly addressed.

[14] The users were divided into a high and a low response group according to their reactions; only the high response group had significant changes

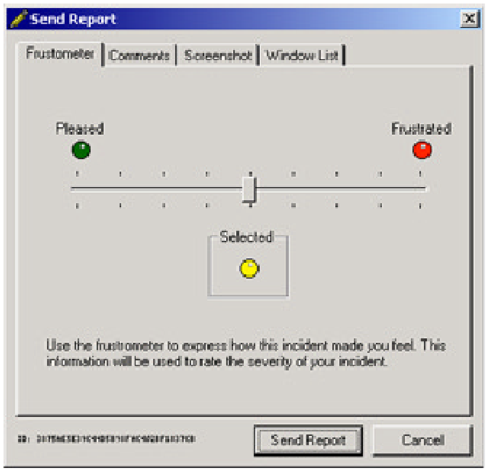
[15] The concept of "adaptive response" in this context is not to be confused with the adaption to a problem being mentioned in Chapter 5.5.

[16] According to Kontogiannis [12], p. 54.

Active support, in contrast, directly addresses the negative feelings and is actively offered by the computer. In “This computer responds to user frustration” [11], Klein et al. showed, that active listening by the computer can significantly increase the willingness of the user to go on working (or, in their specific case, playing). Based on several other research studies, they put together a list of design rules for agents performing active listening. The agent should...

- ...ask the user for his current emotional state in a timely fashion.
- ...allow the user to correct the feedback if the agent misjudged the emotion.
- ...make sure the user can express what he is really feeling.
- ...provide feedback on the emotions and express that the user’s emotional state is valid.
- ...show sympathy and empathy towards the user.

Although the agent designed by Klein et al. was rather simple, as it was based on a simple form with predefined possible inputs and a scripted behavior, it helped the users reducing their frustration by following these guidelines.



Use the frustrometer to express how this incident made you feel. This information will be used to rate the severity of your incident.

Figure 5.3: The Frustrometer by Klein [23]. The users can communicate their level of frustration.

In his thesis “The Sensing and Measurement of Frustration with Computers” [23], Reynolds analyzed different ways of communicating one’s frustration to the computer and showed that users preferred some of them over simple web forms. He got the best results for the Frustrometer (shown in Figure 5.3), where the users could communicate their level of frustration using a slider. The users liked it better than the “SqueezeMouse”, where they had to indicate their level of frustration by squeezing the mouse, as they found the slider easier to use.

5.4 Aggression

There are many different studies regarding aggression against computers - many of them mainly try to figure out the extent of it, some others try to explain it or test theories about it empirically.

One of the earliest one getting bigger attention was “Rage Against the Machine” [19], conducted 1999 by Mori in behalf of Compaq [17]. In a survey among 1250 workers in Britain, around three quarter of the participants said that their colleagues are regularly swearing at their computer and several have seen colleagues bullying their IT department as a result of errors. Among the under 25 year olds, more than one out of six said, they felt like taking out their aggression on someone or something else, and a quarter has seen peers kicking the PC. The study stresses the impact on business, as two out of five participants said that computer problems made them

[17] The original study seems not to be online anymore, however the abstract from Mori’s website is attached to the CD accompanying this thesis.

miss deadlines and one out of ten said that stress created by those problems has caused their professional relationships to suffer. According to Brinks [4], this study also identifies four "rage types" of users:

- **Abusive Annie**: she shows her aggressions as soon as something goes wrong.
- **Controlling Colin**: he is afraid of computers, because he cannot control it. He avoids the computer as much as possible and is supposed to be the rage type with the biggest tendency towards destroying it completely.
- **Analyzing Alan**: he wastes time trying to fix the problem instead of calling the IT department and blames IT afterwards if he misses deadlines as a result of it.
- **Simmering Susan**: she tends to see herself at fault and tries to suppress her aggressions as long as possible.

In "Severity and Impact of Computer User Frustration" [14], 42% of the student and 58% of the workplace users said they were "angry" at the computer after a frustrating experience.

Brinks [4] categorizes acts of computer-related aggressive behavior according to Nolting's taxonomy of aggressions [20]:

- **Statements of displeasure**: cursing at the computer.
- **Aggressions of retribution**: attacking and screaming at the computer as a reaction on malfunctions.
- **Aggressions of defense**: unplugging it, switching it off forcedly.
- **Aggressions of attainment**: threaten the system administrator in order to make him work faster.
- **Spontaneous acts of aggression**: no specific kind of aggression, as it is just done out of habit.

In her thesis "Aggression gegen Computer" [4], Brinks proposes a model for computer aggression and tries to empirically verify it[18]. In course of this study, 340 participants were asked about specific aggressive behavior. The results are shown in Table 5.1.

Did you ever...	**Percentage**
...swear at your computer or a peripheral device?	61,8%
...damage a data medium on purpose?	23,5%
...damage the keyboard?	7,4%
...pull the cables so aggressively, that the plug or connector got damaged?	4,4%
...throw the PC, monitor or other peripheral devices off the table or let them fall purposely?	1,5%
...throw around the mouse or knock it onto the table?	31,2%
...hit the display?	14,7%
...kick the computer-case?	14,7%
I never acted aggressively.	24,4%

Table 5.1: Aggressive behavior against computers; by Brinks [4]. Translated from German.

Some of the hypothesis about this model could not be proven in the procedure of Brink's study; most notably, no significant connection between the frustration and aggressive behavior

[18]A reduced version of it, for practical reasons.

was found. She suspects this is because of suboptimal operationalization of the frustration (an item in the survey, namely “All in all, I feel frustrated by my PC” [19]). She also could not find any significant results regarding differences between men and women. However, younger persons were indeed significantly more aggressive towards computers than older people.

5.5 Adaption

If a malfunction happens more than once and the user cannot solve the problem, then he might adapt to the problem. This means that he either just lives with the problem or finds a way around it. This is, however, not the same as solving the problem, and can in fact be counterproductive.

A very clear example of this was found in the in-depth interviews mentioned in Chapter 4.2. An office worker told, that in his department all PCs take ages (around 15 minutes) to boot up every morning. While this has a technical reason - the documents of the PCs are backed up at the beginning of each session - the delay is often perceived as a major annoyance and malfunction. While the workers in the department cannot change this, many adapted to this problem in one of these two ways:

- Going to drink an additional cup of coffee in the morning after switching on the PC.
- Not switching off the PC in the evening anymore, therefore circumventing the daily backup routine.

It should be obvious that although the second way of adapting to this problem solves the perceived malfunction, it greatly affects the safety of the whole system by preventing the backup routines from running.

5.6 Superstition

While superstition regarding computers as a result of malfunctions seems to be hardly researched yet, there seem to be some hints that this phenomena actually does exist.

Superstition in this context has two aspects: anthropomorphisation and an effect named “placebo-based problem solving” in this thesis.

5.6.1 Anthropomorphisation

Anthropomorphisation refers to the believe that the computer has a will of its own. Although most of the users probably know that computers are just tools, they sometimes act as if the computer was alive. In a web-survey, Brinks [4] asked the users if they “sometimes have the feeling that the PC is crashing purposely to provoke [them]” [20]. 11% said “yes”. 28% claimed that they “sometimes talked with their PC”. This kind of behavior (talking to PCs, giving them names) can also be observed in other situations; it is up to further research to decide the significance of the influence of malfunctions onto anthropomorphisation.

5.6.2 Placebo-based problem solving

Sometimes, in response to malfunctions, actions are performed with the goal of solving the problem, that do not have anything to do with the actual problem. For example, while writing this thesis, a user was observed while having problems with the network: a website did not finish loading. As a reaction, she closed a word processing program running in the background and tried to reload. The website indeed loaded correctly this time. While this was a mere coincidence[21] and there seems to be no logical reason why closing a word processing program could help with

[19] Translated from German: “Alles in allem fühle ich mich von meinem PC frustriert”. Note that the German term “Frustration” is more general than the original English term, as pointed out by Nolting [20].

[20] Translated from German.

[21] Some tests showed, that the website did not load correctly in about a third of all cases, without any recognizable pattern.

network connections, the user believed that “sometimes it seems to help”. While this is but an unproven assumption, it seems, that facing random errors, learning process can occur, which can be exaggeratedly compared to the famous “Superstition in the Pigeon”-experiment [29].

5.7 Pre-studies for the practical part

With two specific applications in mind (which are described more in detail in Chapter 7.2 and 7.3), user-studies were performed to get information about user reactions in two situations of malfunctioning:

- The mouse cursor suddenly freezes, therefore does not move anymore according to the physical movements of the mouse.
- The user wants to use the microphone, although it is muted. The user does not know it is muted and tries to figure out if it is a hardware error, the volume is too low or something else.

5.7.1 Frozen mouse cursor

The reactions of users to this problem were observed using a fake user study about usability of copy/paste handling on a KDE desktop. First, they were asked about their experience with KDE. They were then asked to create a folder and move a given file into it using any of the possible cut- and paste-commands. 8 seconds after they started (the whole task usually takes less than 20 seconds), a C-daemon running in the background grabbed the mouse cursor and made the X-server discard all mouse input. The mouse cursor therefore appeared to be frozen.

Seven students participated at the user study. Immediately after the cursor started to freeze, the following behavior was observed:

- Rapid mouse movements (all participants did it, however for different periods of time).
- Lifting the mouse, rotating it a bit (3 participants, with an angle between 45° and 160°, to see if the (optical) mouse still got power).
- Touching the light sensor (1 participant).
- Putting the mouse onto another surface (1 participant).

In the terms of Chapter 5.1, the first reaction (rapid mouse movements) could be seen as a symptom out of confusion, the other reactions as problem solving. In the terms of Kontogiannis' error recovery processes (Chapter 5.2), lifting the mouse would be part of the error explanation process, putting the mouse onto another surface part of (attempted) error correction, and touching the light sensor probably something in between (the first case would be out of the scope of this taxonomy).

5.7.2 Microphone configuration

The second study was done as an informal survey among six students. They were asked what exactly they did when checking if a microphone was working at all or just adjusted as too quiet.

The reason for this survey was to figure out the best design of an agent trying to detect this specific kind of behavior (in the case that the microphone is muted and therefore malfunctioning from the user's perspective). The assumption was that the common behavior is tapping onto and blowing into the microphone, in order to check if these loud noises can be heard from the loudspeakers. The recognition algorithm could have been optimized in a way to only recognize these two kind of noises.

This assumption was falsified during the survey; in fact, the most common action was to normally speak into the microphone, most commonly “Test 1 2 3”, and either use a headset, or look at a system control panel to see if the computer “sees” noises. Tapping onto the microphone was mentioned by three participants, blowing into it by none at all.

6 Malfunction response

How to respond to malfunctions properly strongly depends on the problem to hand. The possible reactions can be classified by two dimensions: the target of the response and the responsible component in the system.

6.1 Response targets

The target of the system response can be one or more components of the middle column of Figure 3.1: the user, the error symptom, the problem and the manufacturer.

6.1.1 Solving the problem

If the system recognizes the actual problem and is able to solve it, it should do so and communicate it to the user. This is often possible if the problem is due to wrong system settings. However, in many cases it is impossible, especially if the problem occurs because of programming errors in the software, broken hardware or system inherent problems.

For example, the microphone helper project described in Chapter 7.2 solves the problem (the microphone is muted) by adjusting the mixer settings.

6.1.2 Dealing with the symptoms

If the system cannot solve the problem, it can try to reduce the damage and bring the system back into a stable state. Depending on the system design, this sometimes includes some amount of data loss. Therefore, in most cases, there should be additional feedback to the user.

Watchdog timers, checking for deadlocks and system freezes, can be seen as part of this category. They usually only check *if* a deadlock occurred, not *why*. As as response, they do not solve the problem that lead to the deadlock, but only try to bring the system back into a defined, running state.[22]

Another example is the crash recovery of the software package "Open Office" [23], whose crash manager detects crashes of the software, provides feedback to the user and tries to recover the documents being edited at the time of crash.

The mouse helper project described in Chapter 7.3 simulates a reinitialization of the mouse driver that does not work correctly anymore. As this does not fix the actual problem (a programming error or insufficient error handling of the driver), this only cures the symptoms.

6.1.3 Communicating with the user

In almost every case, it will be necessary to communicate with the user. This can serve different purposes:

Keeping the user up to date

Informing the user about what is going on can be necessary for two reasons: from a psychological point of view, it seems to be important for the user to maintain the feeling of being in control - users might feel threatened if the computer is doing too much on its own. On the other side, knowing what is going on helps the user to make the right decisions.

As an extreme example for the second aspect: Kontogiannis [12] argues that an air crash in 1990 happened, because the automatic safety logic masked away the commands of the crew during an engine malfunction without giving them feedback about it, making them misjudge the situation.

In the context of this thesis, which assumes that the user has already noticed the malfunction, the user should be informed if the problem or at least the symptoms are handled, so he knows he

[22]Watchdog timers, however, do not rely on user behavior to detect malfunctions, and are therefore not in the scope of this thesis.

[23]`http://www.openoffice.org/`

can continue to work. Of course, the technical level of detail of this feedback should be considered carefully; while core dumps or blue screens of death certainly are a kind of feedback towards the user, they are not of much help for the majority of users and might not prevent the users from getting aggressive.

Asking the user

When detecting errors and deciding on responses, there is often a high level of uncertainty involved. The two main questions are: Does the user actually see the current situation as a malfunction? And does the found error response lead to a situation that the user perceives as correct?

While the goals of this whole thesis is to help deciding these questions, up to this point (having evaluated the system state and user behavior) there still might remain some doubts by the system. In that case, the system can ask the user about his opinion. Before doing any drastic corrective actions (like closing a program, changing system settings), this also should be done for the reasons stated in the paragraph before.

Prompting the user to do something

The system's range of action is usually limited to operations on a software level and only a handful of operations on a hardware level. However, many malfunctions origin from hardware components that the system does not have any physical influence on.

For example, if the keyboard plug slipped out of the jack, the system cannot solve the problem by itself. In such cases, it is up to the user to perform the corrective response. The system, however, can give him advice what to do and how to do it.

Apologize

In case something really went wrong, the system should apologize to the user. Lewis and Norman argue in "Designing for Error" [15], that this should be done even if an error was provoked by the user, as it should be the computer's task to interpret what the user actually wanted to do. In this context, the system can also give the user chances to vent his frustration (some possibilities are shown in Chapter 5.3.2). The main goal of this is to reduce the stress created within the user when encountering frustrating malfunctions.

For example, Klein et al. showed in "This computer responds to user frustration" [11], that participants played a game with some intentional malfunctions significantly longer if an agent actively listened to the user, apologized for the malfunctions and showed some empathy.

Explain the situation

If there is nothing that can be done about the malfunction, the system might at least try to explain, *why* it is that way. This is, again, to reduce the frustration created by malfunctions.

Two aspects about communicating with the users should be considered: On the one side, asking the user too much and giving him too much information might be annoying for the user and therefore lead to the opposite effect as originally intended. On the other side, when designing the feedback, the situation the user is currently in, should be considered.

The second aspect was fatal in the first iteration of the microphone helper described in Chapter 7.2: the users did not pay attention to the help wizard, as they were busy observing the microphone at that time. The wizard therefore had to be made more flashy than an ordinary wizard, as the attention-getting effect of the window appearing was not effective in this case.

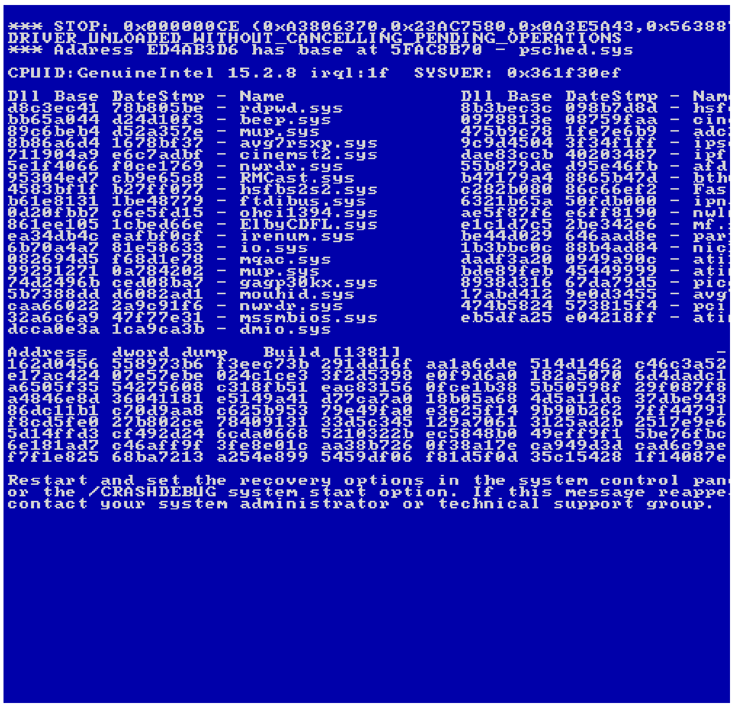
6.1.4 Giving feedback to the manufacturer

In cases where there is an error in a component of a system (document, application, operating system or hardware), it might be helpful to make an error report to the manufacturer of this component. This does not help the user in the short term, but makes appearing problems with a product apparent to the manufacturer, and therefore helps to improve the product[24].

For the manufacturer, system-generated problem-reports have one big advantage over reports by the user: they can include a lot of technical background information that the system might decide not to show the user by default (information provided by a core dump like shown in Figure 6.1 are of little use for the user, but can be a great help for the developers). In contrast, if the user uses a "normal" way of getting in touch with the manufacturer (hotline, web form, etc.), the description will be informal - rather oriented at the "error symptoms", while a system generated error report might reveal more about the underlying problem.

The "Windows Error Reporting" uses this kind of feedback, as an example.

Again, it is important to let the user be in control about what is happening; programs that are "talking home" quietly are seen with a certain amount of suspicion.



```
*** STOP: 0x000000CE (0xA3806370,0x23AC7580,0x0A3E5A43,0x56388
DRIVER_UNLOADED_WITHOUT_CANCELLING_PENDING_OPERATIONS
*** Address ED4AB3D6 has base at 5FAC8B70 - psched.sys

CPUID:GenuineIntel 15.2.8 irql:1f  SYSVER: 0x361f30ef

Dll Base DateStmp - Name                 Dll Base DateStmp - Nam
d8c3ec41 78b805be - rdpwd.sys            8b3bec3c 098b7d8d - hsf
bb65a044 d24d10f3 - beep.sys             0978813e 08759faa - cin
89c6beb4 d52a357e - mup.sys              475b9c78 1fe7e6b9 - adc
8b86a6d4 1678bf37 - avg7rsxp.sys         9c9d4504 3f34f1ff - ips
711904a9 e6c7adbf - cinemst2.sys         dae83ccb 40203487 - ipf
5e1f4066 f0ce1769 - nwrdr.sys            55b879de d95e46fb - afd
95304ed7 cb9e65c8 - RMCast.sys           b47179a4 8865b47d - bth
4583bf1f b27ff077 - hsfbs2s2.sys         c282b080 86c66ef2 - Fas
b61e8131 1be48779 - ftdibus.sys          6321b65a 50fdb000 - ipn.
0d20fbb7 c6e5fd15 - ohci1394.sys         ae5f87f6 e6ff8190 - nwl
861ee105 1cbed66e - ElbyCDFL.sys         e1c1d7c5 2be342e6 - mf.
ea34db4c eafbf0cf - irenum.sys           be44d029 646aad8e - par
6b70a4a7 81e58633 - io.sys               1b3bbc0c 88b4ad84 - nic
082694d5 f68d1e78 - mqac.sys             dadf3a20 0949a90c - ati
99291271 0a784202 - mup.sys              bde89feb 45449999 - ati
74d2496b ced08ba7 - gagp30kx.sys         8938d316 67da79d5 - pic
5b7388dd d6082ad1 - mouhid.sys           17abd412 9e0d3455 - avg
caa66022 2a9c91f6 - nwrdr.sys            474b5824 573815f4 - pci
32a6c6a9 47f77e31 - mssmbios.sys         eb5dfa25 e04218ff - ati
dcca0e3a 1ca9ca3b - dmio.sys

Address  dword dump   Build [1381]                                -
162d0456 558973b6 f3eec73b 291dd16f aa1a6dde 514d1462 c46c3a52
e17ac424 07e57ebe 024c1ce3 3f2d5398 e0f9d6a0 182a5070 6d4dadc1
a6505f35 54275608 c318fb51 eac83156 0fce1b38 5b50598f 29f087f8
a4846e8d 36041181 e5149a41 d77ca7a0 18b05a68 4d5a11dc 37dbe943
86dc11b1 c70d9aa8 c625b953 79e49fa0 e3e25f14 9b90b262 7ff44791
f8cd5fe0 27b802ce 78409131 33d5c345 129a7061 3125ad2b 2517e9e6
5d14ffd3 cf492d24 6cda0668 5210322b ec5848b0 49eff9f1 5be76fbc
6e181ad7 c46aff9f 3fe8e01c aa38b726 0f38a17e ca949d3d cad6c9ae
f7f1e825 68ba7213 a254e899 5459df06 f81d5f0d 35c15428 1f14087e

Restart and set the recovery options in the system control pan
or the /CRASHDEBUG system start option. If this message reappe
contact your system administrator or technical support group.
```

Figure 6.1: A core dump is of little use for the user, but can provide useful information for the developers.

6.2 Components in the system

Regarding the question which component is responsible for handling a malfunction, a layered approach is proposed by this thesis. The layers are similar to the model used in Chapter 4.3 to describe the possible sources for problems. However, as there are problems the system cannot solve on its own, especially hardware related problems, an additional layer for the user is introduced. The layered system for malfunction response can be seen in Fig. 6.2.

Problems in a component can usually only be solved by the component itself, as solving a problem needs much insight into the inner structure. Components usually also cannot do much about malfunctions in lower layers - for example, applications cannot solve problems in the operating system (except if the operating system explicitly allows this use case). So to respond on a malfunction, only the component itself and the components below are involved. This idea is expressed in Figure 6.3. Note that the error handlers and observers are not a real additional layer between two components in the sense that all communication goes through them, but rather a unit of its own that just knows both components and receives information from both, therefore being something in between from a conceptual point of view.

[24]Provided that the manufacturer actually cares about the problem reports.

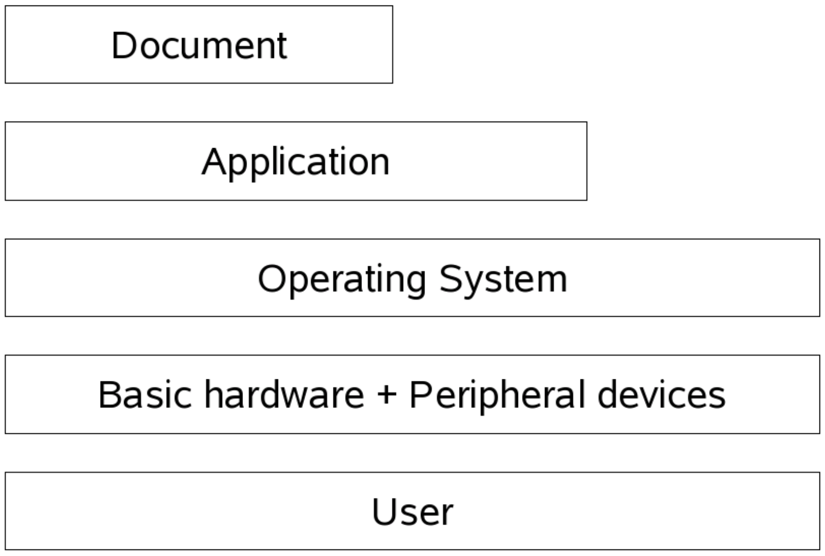


Figure 6.2: Layers involved in malfunction response.

The following sub-chapters explain, what this means in detail in respect to the layers shown in Figure 6.2.

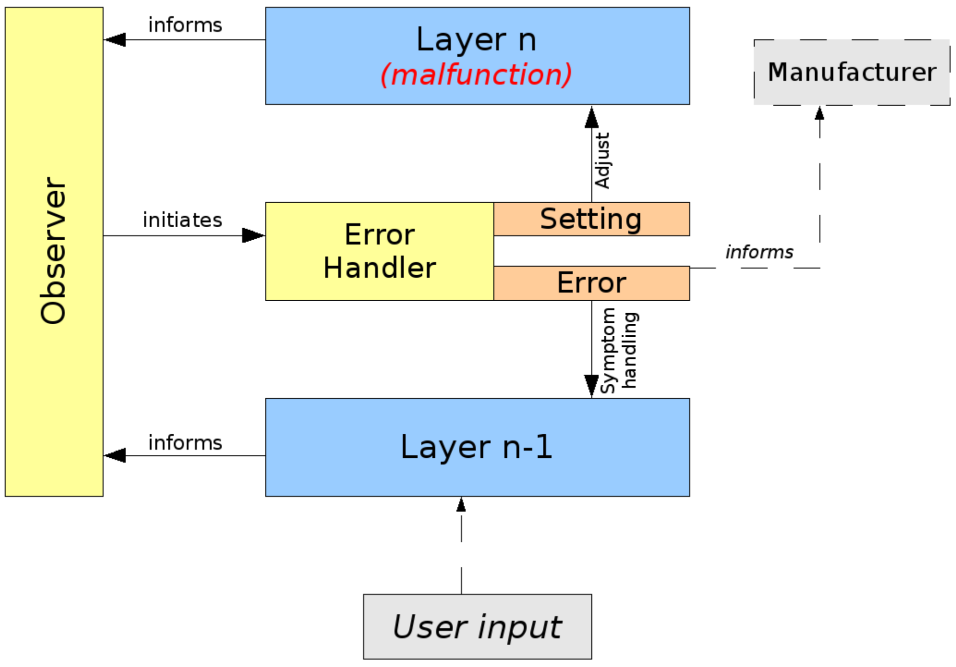


Figure 6.3: Conceptual model of the malfunction response hierarchy.

6.2.1 Problems in the document

Documents (like websites or presentations) usually cannot do much introspective error handling on their own, so the two possibilities are to handle the symptoms (trying to show the parts of the document that are still intact; informing the user about it), and to inform the "manufacturer", the person who created the document.
For example, both is done in the Firefox-extension written for this thesis, which is explained in Chapter 7.4. The extension makes the decision to start error handling based on JavaScript error-messages provided by the Firefox error console (Application layer), mouse movements (reported

by the Application layer) and the DOM-structure of the document (Document layer).

6.2.2 Problems in the application

If the problem is a wrong setting, the application has to adjust the settings itself. If there is a (programming) error in the application, the application probably cannot handle it itself, so the operating system has to perform symptom handling. Triggering the malfunction response can be done by the application itself or the operating system (if the operating system notices that the application is doing something strange).

For example, in another scenario of an erroneous microphone-configuration, the application might listen to the microphone device of the rear microphone jack, while the microphone as actually plugged into the front one. The operating system might notice that a signal is coming from one input, while the application is listening to another one. Based on this observation, the application could be notified to adjustment this setting.

Some applications like Open Office do a certain amount of crash recovery (symptom handling) on their own; in that case, the component of the application which is responsible for crash recovery could conceptually be seen as an addition layer between the main application and the operating system.

6.2.3 Problems in the operating system

The error handling unit gets information from the operating system and the hardware. Again, if the problem is a setting, the operating system can fix it on its own, while most bigger problems like system crashes are more likely to be handled by some hardware components. However, most modern operating systems heavily use a layered approach themselves, so a lot of symptom handling is done by the operation system itself.

Both the microphone helper and the mouse helper fall into this category; the microphone helper needs low-level access to the sound card (hardware layer) and the current sound settings of the operating system and triggers problem solving by adjusting the operating system. The mouse helper gets a signal about hasty movements from the mouse, inferring that the mouse driver of the operating system is malfunctioning and consequently does symptom handling by reinitializing the driver.

6.2.4 Problems in the hardware

Many problems with the hardware will have to be physically solved by the user himself (by plugging in a cable, changing hardware components, resolving a paper jam in the printer, and so on). Feedback to the user usually will be given using the operating system (the layer above), but will be triggered by the hardware.

7 Evaluation of practical projects

Chapters 3 to 6 explained on a rather high level of abstraction the necessary components and concepts for behavior-based malfunction response. To find out if behavior can be used in practice to detect malfunctions, and if such systems have an actual advantage, three systems were developed and two of them empirically evaluated.

7.1 Three different projects

The decision to design three smaller systems instead of one more complex one was made because of the broad field of research to hand. While three systems are still not enough to be truly representative for all possible applications, they cover different typical use cases regarding the type of malfunction, the role of the user reaction and the goal of the system response.

The types of malfunctions are:

- Microphone helper: A perceived malfunction in the layer of the operating system. The observation is done on the operating system layer.
- Mouse helper: A technical malfunction on the layer of the operating system. The observation is done on the hardware layer.
- Frustration 2.0: A technical malfunction / incompatibility at the document layer. The observation is done on the application layer.

The roles of the user behavior are:

- Microphone helper: From the user behavior, an expectation that cannot be met by the current system configuration can be deduced.
- Mouse helper: A behavior pattern typical for confusion triggers error detection and response.
- Frustration 2.0: User behavior is used as a criteria to avoid unnecessary error handling.

The goals of the system responses are:

- Microphone helper: Reducing the time the user needs to fix the problem himself.
- Mouse helper: Obsoleting the need to reboot the system or wait until the problem vanishes by itself.
- Frustration 2.0: Lessen user frustration and provide detailed information to the manufacturer.

Especially in the first two applications the implementation was done prototypically; a final product would need changes in the operating system and the hardware. As the goal in this thesis was rather to get hints on the benefit of such systems than to create the actual system, prototypical implementations with simulated errors are sufficient.

This thesis also does not cover questions regarding the "return of investment" or the organizational practicability (for example, extending communication-protocols between hardware and the operating system) of such systems.

7.2 Microphone Helper

7.2.1 Introduction

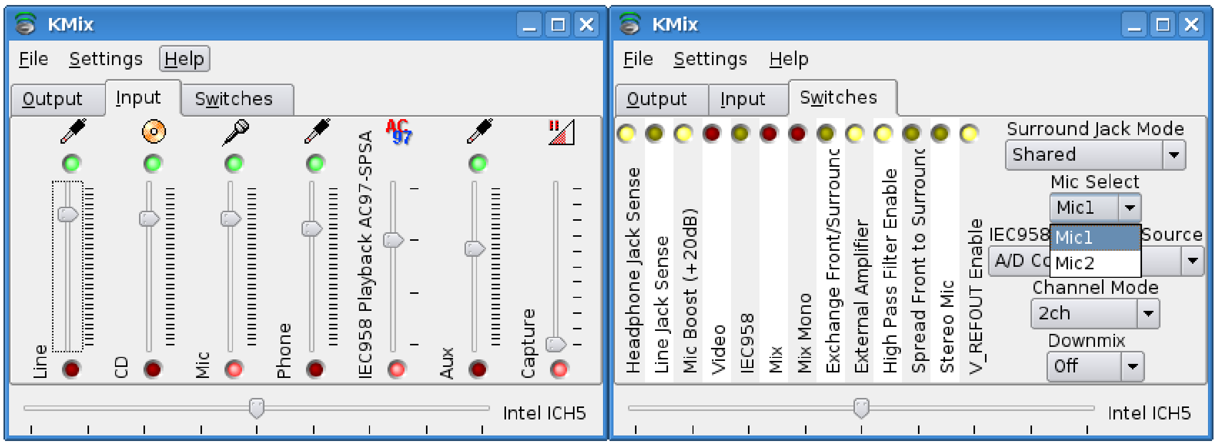
It seems to be a straight-forward task to configure a microphone on a computer. However, in practice, there are several mistakes that can be made when setting it up.

The idea to use the microphone configuration as one of the scenarios to analyze behavior-based malfunction response is based on personal experience made during a university's multimedia-tutorial: both the students and the tutor had a hard time to get the microphone working correctly on the Linux-PCs.

Sources of problems can be:

- The "Capture" flag (bottom right of Figure 7.1a) is not set. In this case, the inputs of the microphone are played back, but not recorded.
- The microphone-specific volume controls (the two toggles and the slider of "Mic" in Figure 7.1a) are not set correctly.
- The wrong microphone is selected ("Mic Select" in Figure 7.1b), if the sound card has more than one microphone jack (for example a microphone jack and the front and the rear of the computer).
- The microphone might just not be plugged in or switched on, have a defective contact or empty battery (for wireless microphones), or any other hardware-related problem.

The microphone configuration in Linux seems to be especially hard: the high flexibility leads to many configuration options, and the configuration tools like "KMix" are not overly user-friendly, as can be seen in Figure 7.1. The fact that many artifacts of the old sound driver system OSS still remain in many systems in order to maintain compatibility to legacy programs makes things even worse.



Surround Jack Mode: Shared
Mic Select: Mic1
Channel Mode: 2ch
Downmix: Off

Figure 7.1: Two views of KDE's mixer application "KMix". While being quite flexible, it is hard to figure out where the problem lies if something does not work. (a) The sound volumes and input selection. (b) Additional switches and modes.

Simple usability engineering probably could improve a lot; the mixer used by Windows XP seems to be easier to use (and a bit less flexible). Baudisch et al. invented the "Flat Volume Control" [2], which improved the usability of the mixer even more and was implemented in Windows Vista[25]. Still, provided that a certain amount of flexibility (and hence chances to mis-configure it) will remain, it is probably too much to hope for that users will never have problems anymore setting up the microphone.

[25] Although it was mainly concerned about simultaneous audio playback of sounds by different applications, rather than the microphone configuration.

In fact, in the user survey of Chapter 4.2, 11% of all participants said they had problems configuring the microphone on a "regular" base, while 14% had them "a few times" (60%: "never", 15%: "only one time"). As no distinction was made regarding the frequency the users actually tried to use the microphone (not only how often they had problems), 60% saying they "never" does not necessarily mean that all of them are actually using the microphone without a problem.

So this thesis tries to find out, if a system that tries to react on user behavior in the case of perceived malfunctioning actually can help in such a situation.

The concrete error situation examined is: the user wants to record something using the microphone, however nothing is recorded, as it is completely muted. As microphones are often muted to avoid humming or acoustic feedback, this appears to be a feasible error situation, especially for users who use the microphone hardly at all (and hence do not know where exactly to unmute the microphone).

To break this case down according to the components introduced in this thesis:

- The **problem** is a wrong setting in the sound system of the operating system.
- The **symptom** is that the program is not recording anything, the microphone is "not working".
- The **user reaction** the system is observing is the problem solving behavior (trying to figure out if the microphone is working or not) - mainly speaking loudly into the microphone. As described in Chapter 5.7, the behavior was validated before implementing the observer by a small user survey.
- **Observation / diagnosis**: an error is assumed if the user is speaking loudly into the microphone while it is muted.
- **System response**: The user is told that the microphone is muted. The system asks if it should solve the problem by adjusting the settings.

To evaluate it, the whole system was simulated; the architecture used for this simulation was considerably different from what an actual implementation for the "real world" would be; therefore, both implementation architectures are explained.

The microphone helper was tested in two iterations - the second one was not planned initially and introduced because of the unsatisfactory results of the first test row. The reason for these results are explained, as they inherently have to do with the nature of the whole system.

The results of both test rows and a control group are presented, along with the results of a survey among the participants.

All tests (including control group) were conducted on the same laptop, using the same microphone. The participants could use an optical mouse, to avoid that the measured times are influenced by the users having to adjust to the laptop's TrackPoint.

7.2.2 Implementation of a real system

The whole system runs in the domain of the operating system. Detecting that the user is speaking into the microphone could be done in the microphone itself or the sound card as well, but given the fact that this would implicate the need of new communication protocols between the hardware and the operating system and is less flexible, this seems to be the less realistic option.

In order to detect it in the operating system, the sound must not be muted on the hardware level (the sound card). The detection is part of the sound driver (for example as an ALSA-module[26]) and introduces one new interface following the observer-pattern to upper layers: if the user is speaks loudly into the microphone while it is muted (or set to a very quiet volume

[26]ALSA is the "Advanced Linux Sound Architecture". `http://www.alsa-project.org/`

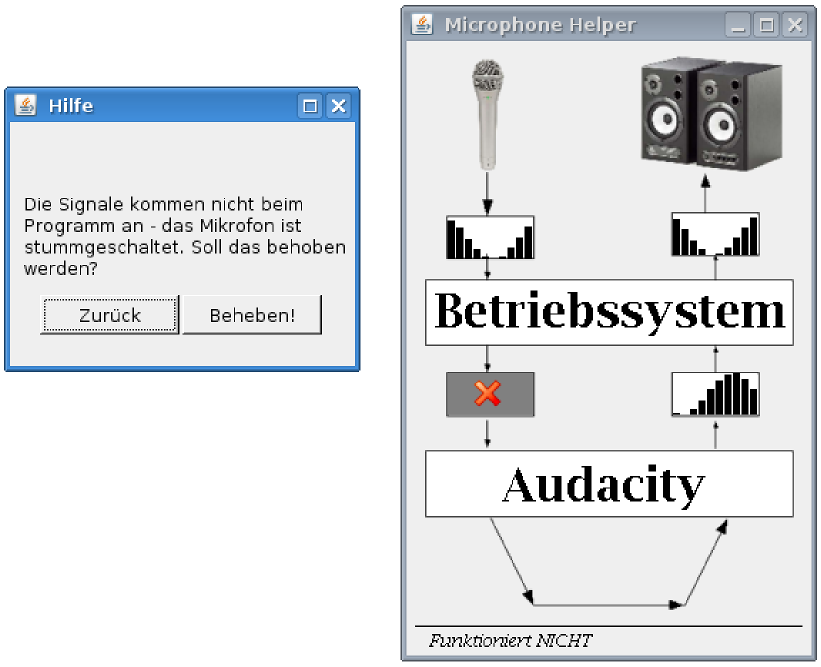
level), it notifies interested processes about it. A part of the desktop system can then handle the situation, ask the user and modify the sound settings according to the user's wishes.

The whole system has a inherent privacy-related problem: on some level, the system has to listen to the user and evaluate the input, even though the user has muted the microphone before. As shown in the survey in Chapter 7.2.7, some users do have problems with this fact, which has to be considered (especially in regard to laptops and other computers with built-in microphones, where the user cannot avoid the problem by just pulling out the plug). So on the one side, the user might be given the chance to deactivate the whole system. On the other side, the system architecture should make sure that if the microphone is muted, no actual sound data can be gained through the sound drivers, except the Boolean value of "is the user currently trying to use it".

7.2.3 Implementation of the evaluation system

The evaluation system was implemented in Java. This brought along one design restriction: even though the sound system ALSA theoretically supports multiple programs sharing one input line (like the microphone), this did not work for two different Java processes using the Java Sound System. As two parts of the system needed the audio data of the microphone - the recording application and the error observer - they had to run in the same process. From a conceptual point of view, this is not very elegant, as the two parts of this process represent totally different layers in the real system (the recording is part of the application layer, while the observer is part of the low-level operating system). To separate these two components at least a bit, they were implemented as two different classes (`MicWatcher` and `RecorderPanel`), both of them implementing the `SoundUpdatable` interface registering at the `RecorderThread` (representing the low-level sound driver). This seemed like an acceptable compromise.

The help wizard (which should be part of the desktop system in a real implementation) was implemented as a completely separate Java-process. Normally it is not visible except for an entry in the system tray, and only listens on a predefined TCP-port. It opens, once some other process connects to this port, and does the further interaction with the user.



Die Signale kommen nicht beim Programm an - das Mikrofon ist stummgeschaltet. Soll das behoben werden?

Figure 7.2: The helper dialog of the first iteration. The red X indicates where the problem lies. By clicking on it and the "solve it" button, the user tells the system to fix the problem.

The help wizard, which is shown in Figure 7.2, was designed in a way to allow the handling of different error situations regarding the microphone configuration:

- No signals arrive at the operating system, indicating a malfunction in the hardware (for example, the microphone is plugged into the wrong jack, its battery is empty, or it is switched off). Observing this kind of situation needs additional monitoring devices, as the system can not rely on the sound input of this microphone anymore. The Always-On Microphone suggested in Chapter 8 might be an approach.

- The signals arrive at the operating system, but not at the application. The implemented error situation falls into this category. Another error situation would be an application listening on the wrong audio device.

- The recording itself works correct, but the user cannot hear the noises as the sound output is not configured correctly. Depending on the concrete error situation, additional means of observation would be necessary as well.

To simulate the error, one setting of the audio configuration was "abused": the microphone was regarded as muted for recording, if the "mute" toggle of the microphone in the input-tab of KMix was switched on. This is, however, not the real behavior: the mute-toggles in the input-tab only affect the immediate output of the input devices, while the recording policy is made by the Capture-toggle (the Capture-slider does not have any visible effect at all). The `RecorderThread` periodically checks the status of this setting (by calling and analyzing the output of the mixer-program `amixer`) and either hands on the sound data unchanged to the recording tool, or muted.

The observing component `MicWatcher` analyzes the unmuted sound input data. If the sound volume is above a predefined threshold for a given time (0.16s) and the microphone is muted at the same time, it is regarded as a malfunction situation and the help wizard is called. More sophisticated recognition algorithms are imaginable to look for more specific behavior of the user; for example, using frequency analysis it is possible to distinguish between the noises produced by tapping onto or blowing into the microphone and speaking into it. A simple implementation of such an algorithm using C and the OpenSource-library FFTW[27] was implemented in the progress of this thesis and can be found on the CD of this thesis. However, as the program is supposed to react on all these behaviors, no distinction was made.

7.2.4 Control group

To have reference data which the results of the helper system can be compared against, a test row was conducted where the users had to solve the problem without using the microphone helper, only relying on the regular sound mixer.

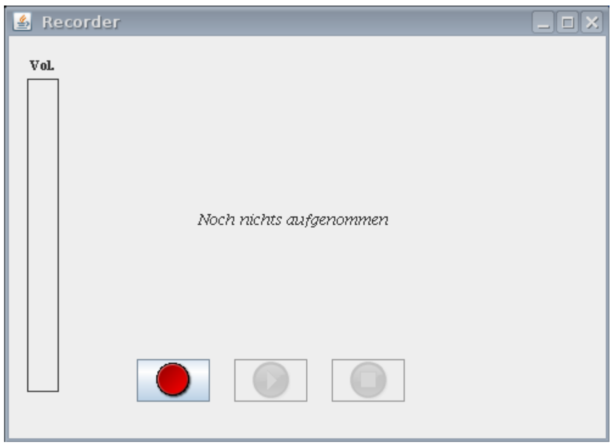


Figure 7.3: The simplistic recording program used for the study.

[27] `http://www.fftw.org/`

13 participants, mainly students, were given the task to plug in a headset, open the recording program (shown in Figure 7.3) and record something (“test 1 2 3” was mentioned as an example). They had to make sure that it was actually recorded. The volume bar at the left of the application window gave an indication whether recording was working correctly. Before the test started, they were shown the icon of the mixer in the system tray and told they could use it to adjust the volume.

The correct way of solving the problem was to open the mixer, open the input-tab and switch on the Capture-toggle. This was regarded as a rather complex problem, as it was predictable that most participants had hardly any experience with KDE’s sound mixer and the Capture-toggle is not very self-evident.

To get an estimate about how long it would have taken the user if the problem to hand was actually an easy one for the user, a trick was applied. Two times were measured per test: the time until the problem was solved (or until the user gave up), and the time until the user performed the first action that had problem-solving character (like putting the microphone plug into another jack, or changing any toggle or slider in the mixer). The measured times were taken as an rough estimate of the time the user would have needed if it had been a trivial problem.

The times needed for the different steps in the task were measured by hand using a self-written JavaME[28]-application on a cell phone. If a participant could not solve it within 150 seconds, the test was canceled.

After the test, the participants were asked to fill out a survey, asking them about their computer experience and if they would like the computer to help them in such situations.

Results

Out of the 13 participants, only 4 could solve the problem at all within the time limit, which results in a solving rate of 30%. Most participants actively gave up before the time limit, as they did not have “any idea anymore”. That is, although the self-estimated computer-experience was quite high (4.33 on a 5-point Likert scale with 1 being “beginner” and 5 being “expert”).

For those four participants who could solve the problem, the average time to solve it after they noticed the problem was 86 seconds (the values ranged from 54 to 115 seconds). The average time for the first problem-solving attempt was 16 seconds (the single values ranged from 7 to 31 seconds).

Although they were explicitly told about the possibility to adjust the volume, most participants searched for an hardware problem first, by plugging the microphone into another microphone jack.

Ironically, the “abuse” of the microphone mute-toggle in the helper system probably would have improved the usability of the whole sound system: in the control group, where the microphone was muted for recording by deactivating the Capture-toggle, most of the participants tried out the microphone mute settings, but hardly anyone noticed that the Capture-button had anything to do with it.

7.2.5 First iteration of the helper

The two main questions of this study were:

- Does the program recognize the behavior of the participants, trying to use the microphone?
- How long do they need to fix the problem using the helper?

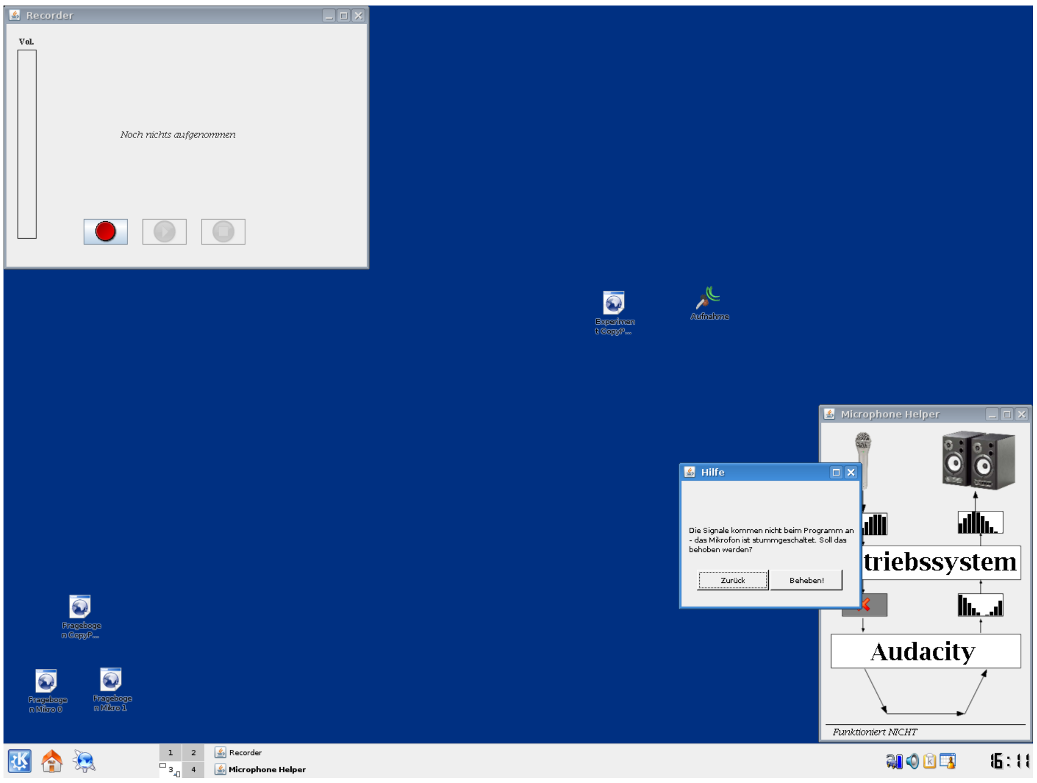
The first implementation of the helper system was tested by 14 participants.

They were given exactly the same instructions as the control group. This time, however, the helper daemon was running in the background and the volume settings were slightly different (the microphone-toggle was muted, not the Capture-toggle, to make the listener program work). As the participants were not supposed to use the mixer in this test anyway, provided that the

[28]Java Platform, Micro Edition: `http://java.sun.com/javame/index.jsp`. For mobile devices.

detection algorithm works correctly, this does not affect the results and is therefore acceptable. If the detection algorithm did not work correctly, the test was regarded as a failure.

Once the participants opened the program and tried to record something although the microphone was muted, the help wizard popped up at the bottom right of the screen. The participants could then solve the problem by clicking on the error-symbol of the image and then confirming the unmuting. The whole desktop including the helper is shown in Figure 7.4.



Die Signale kommen nicht beim Programm an - das Mikrofon ist stummgeschaltet. Soll das behoben werden?

Figure 7.4: The desktop of the test laptop, after the help wizard opened.

Results

The self-estimated computer-experience of the participants was on average 3.36 on a 5-point Likert scale and therefore less than at the control group.

Out of the 14 tests, one was skipped due to technical problems. The user behavior was correctly interpreted in 11 cases and not recognized in 2 cases, which results in a recognition rate of 85%.

Whereas this can be seen as quite successful, the average time the participants needed to solve the problem is disillusioning: 46 seconds with a standard deviation of 30 seconds (the values ranged from 14 to 106 seconds) - for making just two mouse clicks.

This is still much better than the control group; after all, all participants could solve it (compared to only 30% in the control group), and the average solving time was significantly lower ($p = 0.03$) than the 86 seconds of the control group. However, compared to the time of the simulated easy problem, it was significant slower ($p = 0.01$) - and 46 seconds just seem to be too much from a subjective point of view.

As this problem was already noticed during the tests, the participants were informally asked about it after the test and the survey. It appears that most of the users just did not notice that the help wizard popped up at all, and tried to solve the problem by other means (as in the control group, many plugged the microphone into another jack). Only when they tried to change the volume settings, most of them noticed that an additional window was there, got the idea and solved the problem.

This problem of the participants not seeing the help wizard was surprising. The reason for it became clear after observing the chain of actions more closely: the help wizard opens immediately as a response on the user speaking loudly into the microphone. At this time, the user is highly concentrated on either the microphone or the recording application (which is in the upper left of the screen as can be seen in Figure 7.4 and Figure 7.5). This state of being highly concentrated on one thing draws away the attention, therefore the helper's attention-getting effect of appearing out of nowhere was mainly in vain.

That is a finding that generally should be kept in mind when designing systems that respond on such user behavior: when encountering a malfunction and trying to figure out the problem, users tend to be fixated and attention-getting methods that usually work might fail.

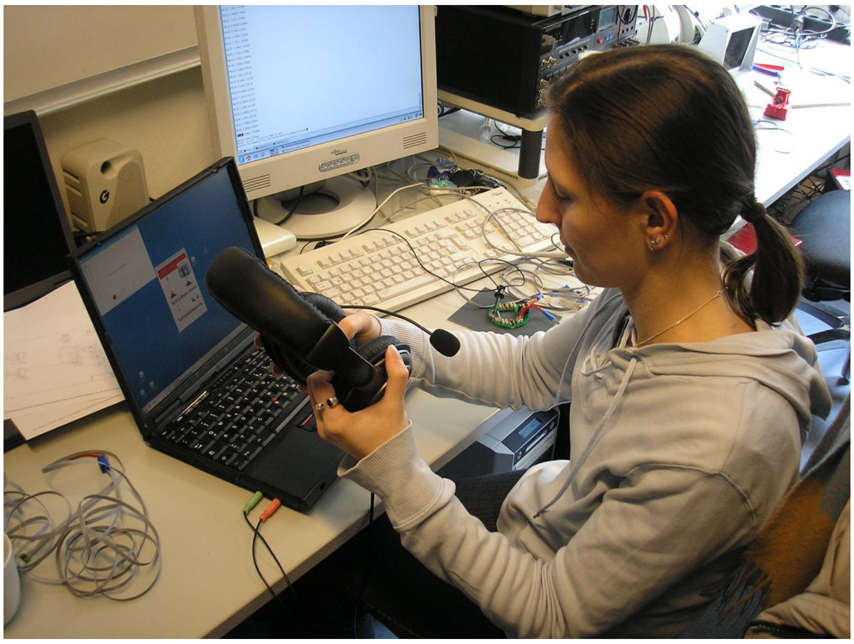


Figure 7.5: The participants were highly concentrated on the headset's microphone, so the help wizard had to be more flashy to get attention.

7.2.6 Second iteration of the helper

With the problems of the first iteration in mind, the help wizard was redesigned for another study. The helper now appeared in the middle of the screen, not in the lower right anymore. An error message using red background was used and a "Solve it"-button was placed right next to the error message. The desktop, including the new helper, is shown in Figure 7.6.

The aesthetic aspects of the second version certainly are suboptimal. Also, considering that in a real implementation there still might be the case of a "false alarm" (for example, the user is tapping onto or speaking right next to the microphone, but actually does not want to use it), an overly obtrusive help wizard like this might be highly annoying. However, designing help wizards for this special situation is not in the scope of this thesis. As the goal of this thesis is to find the possible gains of a error-responsive system, this design seems to be justifiable. Finding better ways to design such help wizards is up to further research.

The test design was the same as at the first iteration. Solving the problem was similar, just with the immediately-visible "solve it"-button, one click could be saved.

Results

12 students participated the test, the self-estimated computer experience was 3.75 on a 5-point Likert scale.

The program reacted correctly in all 12 cases and all participants could solve the problem. The average time to solve the problem was 21 seconds with a standard deviation of 15 seconds (the values ranged from 6 to 54 seconds).

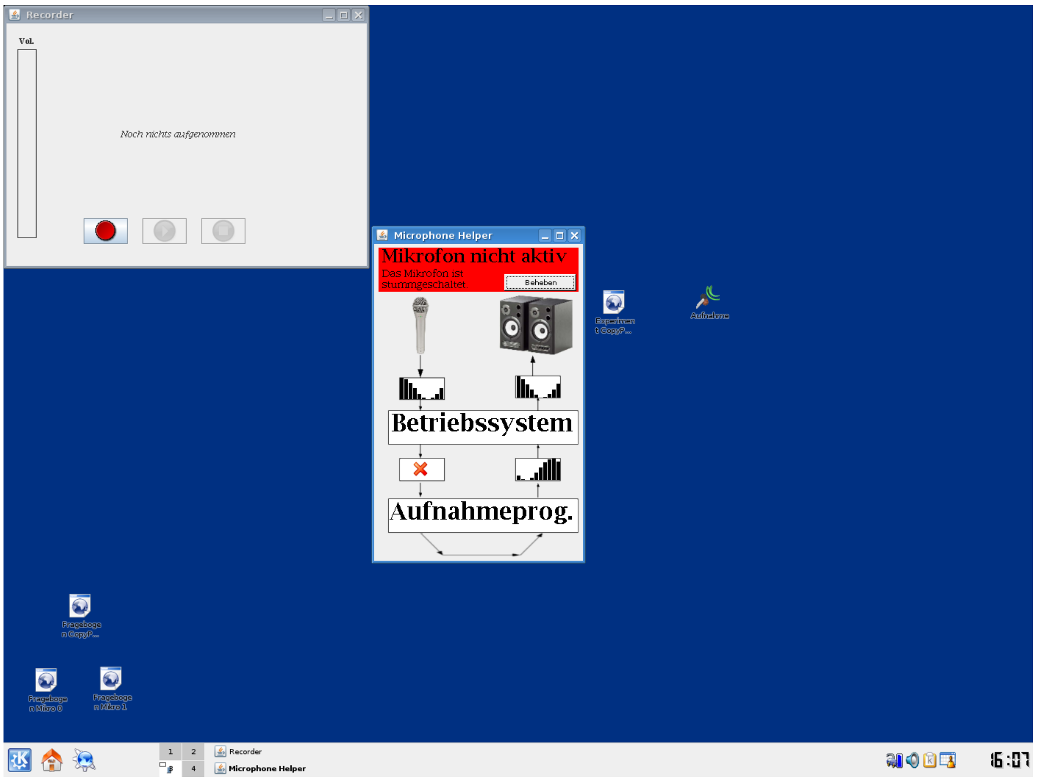


Figure 7.6: The second iteration of the Microphone helper. “Microphone not active” and “solve it” is written with a red background.

The solving time was significantly lower than with the first iteration (p = 0.02), and therefore also significantly lower than the control group (p = 0.00). Still, the average time was above the time needed for the simulated simple problem (16 seconds). However, that difference is not significant (p = 0.31).

7.2.7 Results of the questionnaires

Each test was accompanied by a survey. The questionnaires for the two iterations of the microphone helper were identical, the questionnaire for the control group slightly different: the questions regarding the microphone helper were skipped, the other questions were rephrased to fit the situation. The exact questions can be found in Appendix A.2.

The questions shown in Table 7.1 had to be answered on a 5-point Likert scale with 1 being “completely disagree” and 5 being “completely agree”. The average values and standard deviations are shown in Table 7.1.

Unfortunately, it appears that a couple of answers by the participants were not recorded - either some participants forgot to press the “send”-button, or, more likely, the data were not saved correctly. It appears this happened more or less randomly, therefore it should not affect the Representativeness too much.

The most interesting item in the results of the questionnaire is probably the question “I prefer to solve such problems myself”. While the average (3.11) is close to the middle, the standard deviation (1.55) shows that there the users actually have completely different and mostly clear opinions about it - Figure 7.7 illustrates this.

So while one group of participants prefers to solve problems themselves, another group clearly prefers not to. To see if there are further differences between these two groups, the other results were divided into them. The first group SELFSOLVE was defined by those participants answering with 4 or 5, the second group NOTSELF by those answering with 1 or 2. The average results for these two groups are listed in Table 7.2.

As could be expected, the members of SELFSOLVE are more experienced with computers.

	Average	**Std. Dev.**
I prefer to solve such problems myself	3.11	1.55
If I had to solve this problem myself, it would have taken me longer[29]	3.89	1.24
I'm bothered by the thought that the computer would always "listen", even if the microphone is muted	2.37	1.31
Altogether I'd welcome if the computer tries to act on its own, once I have problems	3.44	1.09
I had a similar problem before	2.85	0.95

Table 7.1: Questions that were asked in the survey about the Microphone helper.

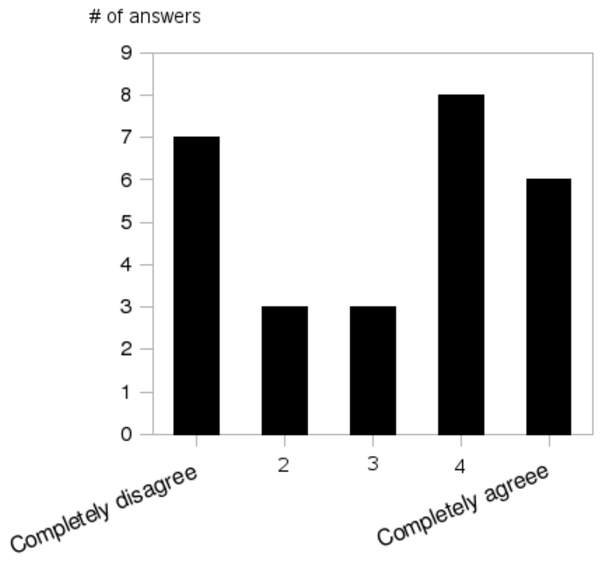


Figure 7.7: "I prefer to solve such problems myself" - two distinct groups.

They are more likely to be bothered by the thought of the computer "listening" and are less convinced of the advantage of the microphone helper system. However, this is not confirmed by the actual measured times: the average time to solve the problem (including both iterations) was 38 seconds for SelfSolve, compared to 31 seconds for NotSelf.

Comparing the answers of the users of the second iteration with the control group, there was one major difference: the control group, who had to solve the problem on their own (and mostly failed), still preferred significantly ($p = 0.03$) more to solve problems on their own (average: 4.17) than the group of the users (average: 2.38). The helper-users were also on average less bothered by the thought of the computer "listening", however not significantly.

It should be pointed out, however, that these findings are based on a relatively low number of results ($n = 27$). While most of the results seem to be plausible, they should be confirmed and further analyzed in a bigger survey.

7.3 Mouse Helper

7.3.1 Introduction

In the qualitative survey of Chapter 4.2, the components that were mentioned most frequently as a source of problem, are the mouse and the keyboard. The reason for this probably does not lie in bad quality of the used mice and keyboards, but rather in the fact that they are the

	SelfSolve	NotSelf
If I had to solve this problem myself, it would have taken me longer[30]	3.0	4.5
I'm bothered by the thought that the computer would always "listen", even if the microphone is muted	2.57	1.7
Altogether I'd welcome if the computer tries to act on its own, once I have problems	3.57	3.4
I had a similar problem before	2.93	2.9
Computer-experience	4.0	3.3

Table 7.2: Average values for participants who preferred to solve problem for themselves and participants who preferred not to.

most frequently used components when interacting with computers. Still, as many malfunctions occur at this point, the user-friendliness of computer systems might be improved by adding more malfunction response.

Based on this idea, another application of behavior-based malfunction response was developed, which handles one specific malfunction situation: a mouse driver in the operating system crashed, the movements of the mouse are therefore not (or incompletely) delivered to the upper layers of the system anymore, making the mouse cursor freeze. From the survey and personal experience, the situation where the cursor freezes only temporary and returns to normal after certain amount of time seems to be the more common case. However this assumption is not empirically verified.

The user reactions on this kind of problem were observed in a user-study, which is described in Chapter 5.7. Rapid mouse movements were the common behavior made by all participants. Additional behavior patterns, that were made only by a part of the participants, includes lifting the mouse and moving it to another surface. As for detecting the malfunction reliably, rapid mouse movements seemed to be the best criteria.

If this kind of behavior is detected, additional checks might be made to ensure it is because of a malfunction and not because of some action game the user is playing. If there actually is a malfunction, the system has to decide if it can solve the problem, for example by reinitializing the driver. In any case, it should give feedback to the user.

To break this case down according to the components introduced in this thesis:

- The **problem** is a deadlock or a similar problem in the mouse driver.
- The **symptom** is that the mouse cursor freezes.
- The **user reaction** are the rapid mouse movements.
- The **observation** and diagnosis is done partly on the hardware, partly in the operating system (this is further elaborated in the following chapter).
- The **system response** is to ask the user if the mouse drivers should be reinitialized, and to do so, if the user agrees.

The malfunction, the observation of the user behavior and the system response was simulated and tested in a user study. As this is a problem that users cannot solve themselves (except by rebooting the whole system), it makes little sense to measure the time needed to solve the problem and compare it to a control group like at the microphone helper. More emphasis was put into testing how well the detection of the user behavior works. The simulation of the malfunction in the user study was masked by a study regarding the performance of cut- and paste operations.

7.3.2 Implementation of a real system

The major architectural difference of the mouse system to the microphone helper is the location of the observation component. In Chapter 7.2.2, it is discussed why the observation of the microphone would be part of the operating system, even if it would be technically possible another way. Here, it is the other way around, as the nature of the problem is different: in the microphone's case, everything was working well from a technical point of view and the perceived malfunction was just because of wrong settings. Here, there actually is a technical error going on in the domain of the operating system, so the observation is also impaired on this level - the mouse driver cannot take over the role of the observer.

Theoretically, it would be possible to put the observer into the operating system, on a lower layer than the actual driver. This would require this new component to be a basic mouse driver itself, as it has to interpret the motion events of the mouse. As there is little reason to assume that this second driver would be any more stable than the main one having the problems right now, this way does not look like being very reliable.

The better solution would be to put the observation into the hardware layer, below the operating system. This means, the mouse itself had to reflect on its movements and figure out if the user is behaving according to given criteria. As most mice - especially optical ones - already do a lot of computing internally, this additional computation could be done with reasonable efforts.

This implies that a channel for signaling is required between the mouse and the operating system - to be precise, between the mouse and a error-handling component in the operating system that is conceptionally below the mouse driver, attached to the USB-driver (assuming that we are dealing with a USB-mouse).

While this might appear to be way too much efforts for this one application (after all, the crashing driver is only one possible reason for the often-mentioned symptom of the frozen mouse cursor), a more general way of signaling error statuses and meta-data about user behavior between peripheral devices and the operating system can be helpful in many other applications as well. Just to mention two:

- The keyboard can indicate that the user is currently pressing the keys conspicuously hard or was just knocking onto the keyboard. This might indicate that the user is frustrated right now.
- A notification if a cable is plugged in and pulled out again repeatedly (possibly in a different jack). This might indicate that the user has problems configuring a device (like a printer, sound card or any USB-device).

For the mouse itself, several other error signals could be helpful. Examples of error or alert signals are:

- A signal if an optical mouse detects too much dirt on the light sensor.
- A signal once the battery of a wireless mouse runs low.
- A signal if the surface, the mouse is moved on, can not accurately be tracked by an optical mouse.

Most of these mouse signals would be directed towards the normal mouse driver. The actual protocol design to exchange such meta-data is out of the scope of this thesis. However, it should be pointed out, that the benefits of such a protocol could exceed the field of behavior-based malfunction response.

The rest of the system would be similar to the microphone system: once the error-handling component is reported about this specific user behavior, it makes additional checks to avoid false alarms; this could be done by checking the operation system's internal mouse movements of the last one or two seconds. In case it followed the rapid mouse movements, this could either be normal behavior by the user (for example if he plays an action game or is just playing around with the mouse cursor), or the problem lies in a higher layer (for example, the graphic rendering component does not update the mouse cursor's position). If a malfunction is assumed, some user feedback is given, including the question whether to reinitialize the driver. This feedback has to be designed with special attention, as the user might not be able to use the mouse due to the nature of the malfunction and has to use the keyboard instead. In case the user agrees, appropriate counter-measures are initiated.

7.3.3 Implementation of the evaluation system

The evaluation system does not actually simulate a driver problem, but rather imitates the error symptoms, namely the frozen mouse cursor.

This is done by a small C-program utilizing the X11-API. It repeatedly queries the position of the mouse cursor. As soon as a given file exists (`/tmp/staticmouse`), it repeatedly warps back the mouse cursor immediately (using `XwarpPointer`).

The observation of the user behavior is done in the same program; again, this is not very elegant from a conceptual point of view, but the easiest way to accomplish the task without having to do rather complex access coordination on the X11 event stack between the two components. The program therefore internally keeps two mouse coordinates: the coordinates of the visible cursor, which are static, and the would-be coordinates which are used for analyzing the user behavior.

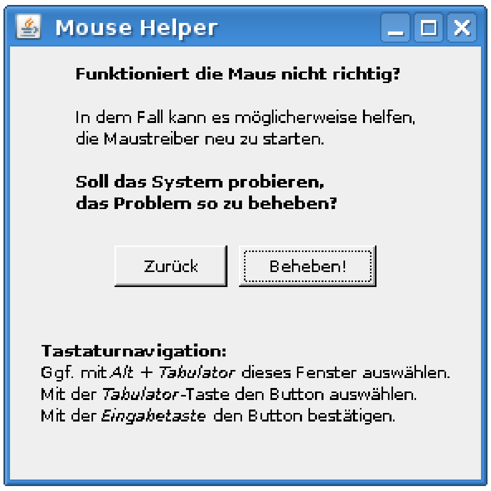
The recognition algorithm is easy to comprehend: it follows the x-coordinates of the mouse cursor movements (the y-coordinates are ignored, as the typical movement of the users was to move the mouse to the left and right or in circles, but not up and down) and records the local minima and maxima of the values. If they are too closely together (less than 40 pixels of difference), they are ignored, as very small mouse movements are rather unlikely in the situation of confused, rapid movements. If the number of minima and maxima in a given time exceed a given threshold (more than 7 within 1.5 seconds), error recovery is triggered. The program code of the recognition algorithm (minus some debug information) is shown in Figure 7.8.

```c
#define MAXIMA_MSECONDS 1500
#define MAXIMA_MIN 7
#define MAXIMA_MIN_XDIFF 40

int maxima_pos[MAXIMA_BUF];
int maxima_msec[MAXIMA_BUF];
int maxima_max = 0;
int last_x = 0;
int last_y = 0;
int direction_up = 0;

void maximumCleanup() {
  int i, j;
  int min = currtime() - MAXIMA_MSECONDS;
  for (i = 0; i < maxima_max; i++) {
    if (maxima_msec[i] <= min) {
      for (j = i; j < maxima_max; j++) {
        maxima_msec[j] = maxima_msec[j+1];
        maxima_pos[j] = maxima_pos[j+1];
      }
      maxima_max--;
      i--;
    }
  }
}
```

```c
void addMaximum(int x) {
  maximumCleanup();
  if (maxima_max > 0) {
    int diff = maxima_pos[maxima_max-1] - x;
    if (diff < 0) diff *= -1;
    if (diff < MAXIMA_MIN_XDIFF) return;
  }
  maxima_pos[maxima_max] = x;
  maxima_msec[maxima_max] = currtime();
  maxima_max++;
  if (maxima_max > MAXIMA_MIN) do_dialog_open_remote();
}

void evaluatemovement(int x, int y) {
  int m_x = (x - last_x);
  if (
    (m_x > 0 && direction_up == -1) ||
    (m_x < 0 && direction_up == 1)
  ) addMaximum(x);
  if (m_x > 0) direction_up = 1;
  if (m_x < 0) direction_up = -1;
  last_x = x;
}
```

Figure 7.8: The C-algorithm to recognize rapid mouse movements.

Error recovery is done by a Java-based GUI application shown in Figure 7.9. It is triggered the same way as the microphone help wizard (and is, in fact, implemented in the same program), by connecting to a predefined TCP-port. It opens a window, telling the user that the mouse driver is not working correctly, and asks him whether the system should try to solve the problem. It tries to fetch the focus of the window manager and sets the internal focus on the “Solve it”-Button, so the user only has to press the Enter- or Space-Key to confirm. In case the focus is not set correctly, instructions how to focus on this window using the (Alt-)Tab keys are printed. In the end, the program just removes the `/tmp/staticmouse`-File, making the C-program stop the error simulation.



Funktioniert die Maus nicht richtig?

In dem Fall kann es möglicherweise helfen,
die Maustreiber neu zu starten.

Soll das System probieren,
das Problem so zu beheben?

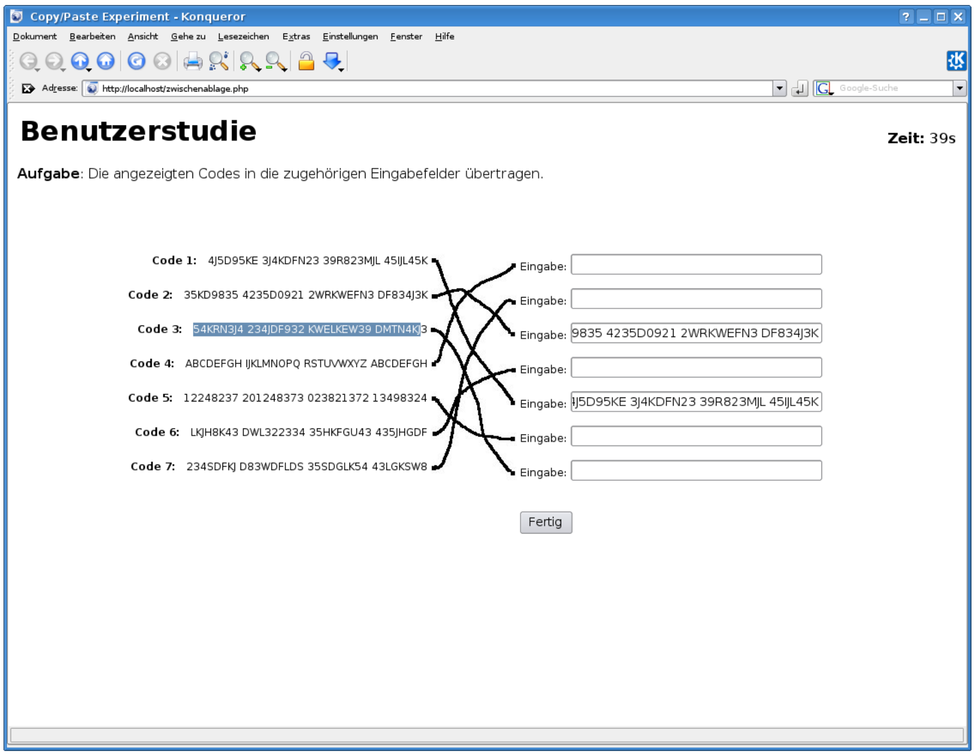
Tastaturnavigation:
Ggf. mit *Alt* + *Tabulator* dieses Fenster auswählen.
Mit der *Tabulator*-Taste den Button auswählen.
Mit der *Eingabetaste* den Button bestätigen.

Figure 7.9: Mouse helper: "Does the mouse not work correctly? [...] Should the system try to fix the problem? [Back] [Solve it].

7.3.4 Evaluation

The main goal of the user study was to get quantitative results about the hit rate of the recognition algorithm - which implies that in fact most of the users show this specific kind of behavior in this situation.

To get authentic results, it is necessary to mask the test, as the behavior to be observed is mainly behavior in response to surprise and confusion. This was done by embedding the malfunction situation in a "fake" user study: the participants were shown a web form (that can be seen in Figure 7.10) with lengthy codes to the left and input fields on the right side. The pretended goal of the user-study was to measure the time the users needed to copy the codes into the correct input boxes using copy/paste. To make things a little bit more complicated, the codes and input fields were connected by random lines crossing each other and thus requiring a bit of attention in order to get the right field.


Copy/Paste Experiment - Konqueror
Dokument Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe
Google-Suche
Fertig

Adresse: http://localhost/zwischenablage.php

Benutzerstudie

Zeit: 39s

Aufgabe: Die angezeigten Codes in die zugehörigen Eingabefelder übertragen.

Code 1:	4J5D95KE 3J4KDFN23 39R823MJL 45IJL45K	Eingabe:	
Code 2:	35KD9835 4235D0921 2WRKWEFN3 DF834J3K	Eingabe:	
Code 3:	54KRN3J4 234JDF932 KWELKEW39 DMTN4KJ3	Eingabe:	9835 4235D0921 2WRKWEFN3 DF834J3K
Code 4:	ABCDEFGH IJKLMNOPQ RSTUVWXYZ ABCDEFGH	Eingabe:	
Code 5:	12248237 201248373 023821372 13498324	Eingabe:	4J5D95KE 3J4KDFN23 39R823MJL 45IJL45K
Code 6:	LKJH8K43 DWL322334 35HKFGU43 435JHGDF	Eingabe:	
Code 7:	234SDFKJ D83WDFLDS 35SDGLK54 43LGKSW8	Eingabe:	

Figure 7.10: The mouse malfunction was masked in a simple user study.

Besides the instructions to copy the values, they were only told that the time would be measured (a JavaScript-implemented stopwatch was displayed on the page to improve authenticity), so they should should work somewhat efficient, but not hastily.

The error occurred when the participants were copying the fifth code: the mouseUp-event of the HTML-document triggers an Ajax-Call to a PHP-script, which creates the `/tmp/staticmouse`-file that tells the daemon to start the error simulation[31].

All tests were conducted on the same laptop with an attached optical mouse.

If the system is working, the participant would do rapid mouse movements once the cursor freezes, the system would recognize this behavior and open a help dialog. The participant would then use the help to let the system solve the problem. If the help dialog does not open within 10 seconds after the problem occurs, the test was regarded as a failure. This time limit was set, because once the short effect of surprise was gone, the participant probably would have realized that the malfunction was part of the test and not act naturally anymore.

After the test, the participants were told that the system was monitoring their mouse movements and the help dialog was shown in response to the rapid mouse movements. They were then asked to fill in a survey.

7.3.5 Results

16 students participated at the user study. The self-reported computer experience was 3.53 on a 5-point Likert scale (with 1 being “beginner” and 5 being “expert”). The computer experience should have less influence on the results of this test, as only the impulsive reactions were tested, not problem-solving behavior.

In one case, the simulation of the malfunction did not work properly (the mouse-event in the browser was not triggered, so the simulation was not started at all), so the test case was discarded.

Out of the remaining 15 cases, the user behavior following the malfunction was correctly recognized in 13 cases, resulting in a success rate of 87%. In one of the unsuccessful cases, the dialog did not show up at all (the mouse movements were too short), in the other case, it did show up, but after more than the time limit of 10 seconds.

The results of the survey show, that while the problem of the frozen mouse cursor is probably not the most frequent problem at all, it did happen to several participants before (see Figure 7.11). About half of the participants who had this problem sometimes before (3 or 4 on the 5-point Likert scale) had to reboot the computer in order to solve the problem. For the other half, the problem disappeared by itself after a while.

Being asked if they preferred to solve such problems alone, the average was again in the middle (3.13 on a 5-point Likert scale with 1 being “completely disagree” and 5 being “completely agree”). However the standard deviation was lower than at the microphone helper (here: 1.13) and no two distinct groups could be identified. This suggests that the middle value was rather due to an indecisiveness regarding the question.

Regarding the privacy aspect of monitoring the mouse movements, there was a tendency towards not feeling bothered, with an average of 2.53 (standard deviation: 1.06). Most participants were rather positive about the general idea of the computer reacting on its own in case of a malfunction (average value: 3.47, standard deviation: 0.99).

The hit rate of 87% in the case of a malfunctions appears to be very good for a first approach. To get an estimate of the overall hit-rate, however, it is also necessary to measure the number of false alarms. This was only done in a very limited way in the course of this thesis: the mouse observer was running in the background for about 30 hours of work and logged the number of

[31] This detour using the PHP-script is needed due to the security system of JavaScript, preventing the scripts to create files on their own.

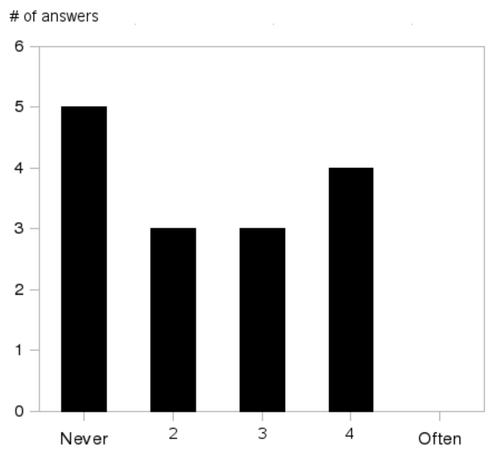


Figure 7.11: "I had a similar problem before" - it is not the most common problem, but some users do encounter it occasionally.

times it would have triggered error handling. As the mouse was working correctly during this time, each error handling attempt can be regarded as false alert.

The result of this small test was that the program would have triggered 2 false alerts in these 30 hours. Whereas this is not very high for a first approach, it would probably be too high to deploy in in a real-world scenario: with a false alert every couple of days, the number of false alerts would probably exceed the number of times the actual malfunction appears, making the Mouse helper more of a nuisance than a help. An approach using OS-internal reference movements (as mentioned in Chapter 7.3.2) and more elaborate recognition algorithms might improve the overall hit rate.

7.4 Frustration 2.0

7.4.1 Introduction

Both the mouse helper and the microphone helper deal with problems in the operating system layer, hints on handling problems on the hardware layer are given in these chapters as well. The third project in this thesis targets a completely different class of problems: problems originating from the document layer, or from incompatibilities between the document and the application layer. Especially due to the spread of the Internet and web applications, problems on this layer have become much more common in the last couple of years.

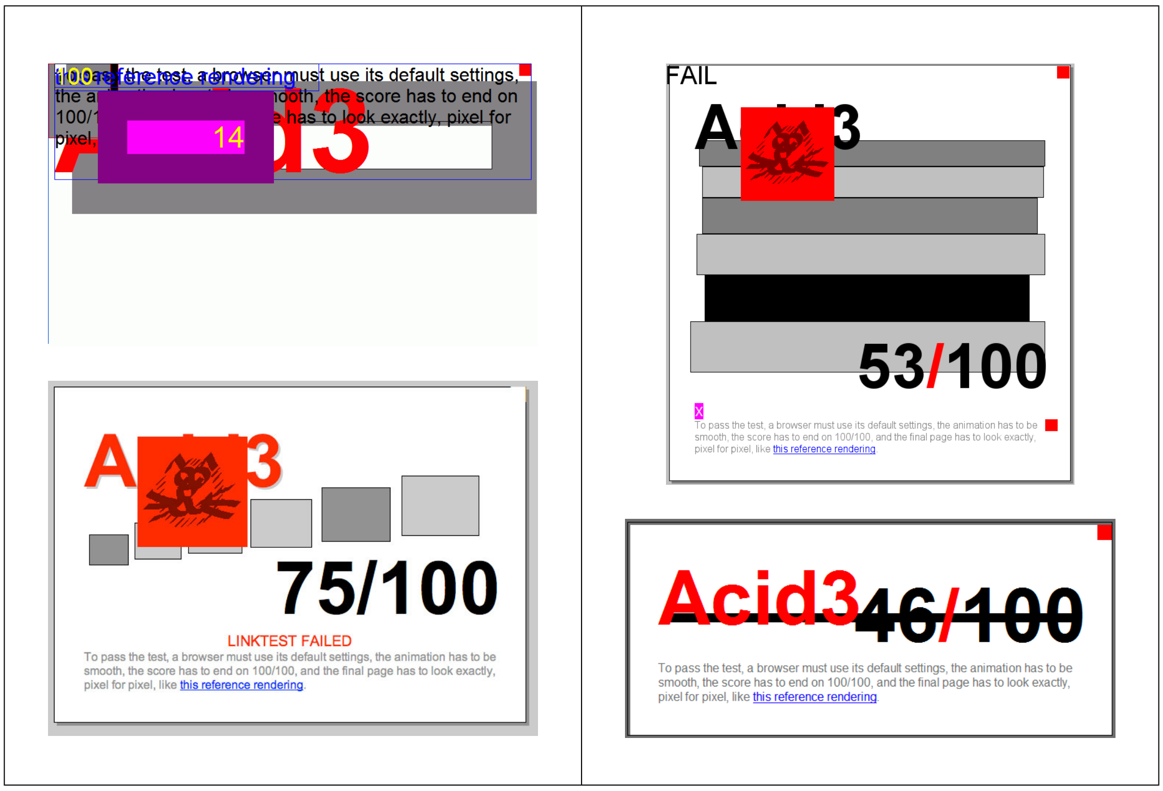
The first peak of problems originating from document incompatibilities was probably at the time of the browser wars between Netscape Navigator and the Internet Explorer [32]. While this has stalled during the following time of Internet Explorer's dominance, it appears this has become a problem again over the last few years for the following reasons:

- New browsers appeared (Firefox, Opera and WebKit-based browsers). Different browsers might render the same site differently. An extreme example can be seen in Figure 7.12.
- The boom of JavaScript-based "Web 2.0"- or "Ajax"-applications and Rich Internet Applications, which lead to an overwhelming development progress regarding active contents.
- Integration of browser engines into other software, re-branding of browsers (like the T-Online browser, which utilizes the engine of Internet Explorer but introduces some new inconsistencies), security software interfering with browser internals (like virus scanners filtering out some cookies or preventing malicious-looking JavaScripts from execution) and pop-up-blockers that sometimes filter out too much.

[32]`http://en.wikipedia.org/wiki/Browser_wars`

Compared to traditional software, websites and -applications have a very short software development life cycle. New features are often added on the fly with little testing, if any at all. This frequently leads to rather frustrating experiences for the users, especially if they are using alternative browsers, because features that worked before might suddenly stop working without prior notice.

The most urgent countermeasures for the current situation would be to improve standard compatibility for browsers and to promote more systematical development processes for web developers[33]. However, behavior-based malfunction response might be an additional approach to help lessen frustration for the user, provided that a perfect world without malfunctions is not about to come along for the Internet world very soon either.



To pass the test, a browser must use its default settings, the animation has to be smooth, the score has to end on 100/100, and the final page has to look exactly, pixel for pixel, like this reference rendering

Figure 7.12: Besides its original purpose as a benchmark for standard compliancy, the Acid3 test (`http://acid3.acidtests.org/`) also demonstrates how differently current browsers might render the same file in extreme cases: Internet Explorer 7 (top left), Firefox 2.0.0.13 (top right), Safari 3 (bottom left), Opera 9.25 (bottom right). [Pictures are Public Domain, from Wikipedia]

For this thesis, the browser Firefox was extended with a behavior-based malfunction response system. This was implemented as an extension and allows websites to add more graceful error handling with little effort and without breaking compatibility to other browsers. The implementation will be explained in the remainder of this chapter. Due to time constraints of this thesis, the benefit of this system could not be empirically verified in time. Measuring the effects on user frustration in the case of a malfunction are up to further research.

7.4.2 Idea of the Firefox-extension

A major problem in handling JavaScript errors is that there are just way too many of them. Looking into the error console of a browser after browsing over a couple of sites reveals how often errors are thrown, even if the user does not actively notice their occurrence. In a way, this is the

[33] At least for the first one, a lot of progress is currently made with the new major browser versions coming up in mid-2008.

opposite of the concept of a “perceived malfunction”: malfunctions that objectively did happen, but are not perceived as such by the user.

This is the reason why it seems to be necessary to look at the user behavior when doing malfunction handling: If a warning or error handler were to appear every time a JavaScript-error occurs[34], it would be highly annoying for the user. A better approach would be trying to figure out if the user actually perceives a given malfunction as such, and only start error handling in that case. The behavior of the user therefore has to be correlated with errors thrown by the browser’s JavaScript-engine.

The current system looks at the case where the user wants to use active content on the website, like a drop-down menu. To do this, he either clicks on it or moves the cursor over it (some drop-down menus open once the cursor touches the parent element). If the user clicks on it and an error is thrown right after this, it should be safe to assume that the error is a reaction on this click and the user notices the malfunctioning.

If the user just moves the cursor over an element and an error is thrown right after, it is not as easy: after all, the user is usually moving the mouse around quite a lot (compared to the frequency of clicking onto something, which is rather low), so if a random error occurs that the user does not notice, it is still somewhat likely that the user just moved the cursor by coincidence. The assumption made in this thesis to face this problem is, that if the user expects a certain behavior after he moves the cursor over a certain element but this behavior is not shown due to a malfunction, he will try it again by moving the cursor away and moving it over the element a second time. If an error occurs again, the likelihood for it being a coincidence is low enough. This assumption, however, has not yet been empirically confirmed.

While the browser cannot fix the problem as a response, there are two things that can be done: apologizing at the user, trying to lessen frustration, and informing the manufacturer (after confirmation by the user).

To break this case down according to the components introduced in this thesis:

- The **problem** is in the document layer: a JavaScript that is either completely broken or incompatible with the current browser.
- The **symptom** is that a specific function on a website is not working.
- The **user reaction** is either to click on an element, or to repeatedly move the cursor over it.
- The **observation** and diagnosis is done on the application layer, namely by the browser.
- The **system response** is to apologize to the user and inform the manufacturer about the problem.

7.4.3 Informing the manufacturer

Informing the manufacturer (in this case, the webmaster) about the problems, is more an advantage for the manufacturer as for the user. Although the user might have an advantage from improved error reporting by faster bug fixes, the main benefit is to reduce the time needed by the customer support to get detailed information about the underlying problem. By themselves, users might skip making a bug report at all, and even if they do, most would probably tell more about the symptoms of the problem than about the problem itself. This often leads to unnecessary long dialogs like shown in Figure 7.13.

It is easy to see that the interaction can be done more efficiently than that. If the browser can send the error message itself to the manufacturer, then there would be no need to explain the user how he can find the error message in the first place. The whole process can partly be automated, although user interaction is still advisable:

[34] Earlier versions of the Internet Explorer actually did that in certain configurations.

User: The pop-up menu is not working anymore.
Customer support: Maybe you're using an unsupported browser? Which one do you have?
User: Firefox 2.
Customer support: That one should work. What error message is shown?
User: Where can I see that?
Customer support: It's in the "Tools"-menu, the item "Error console".
User: Okay, the error is *[error message]*.

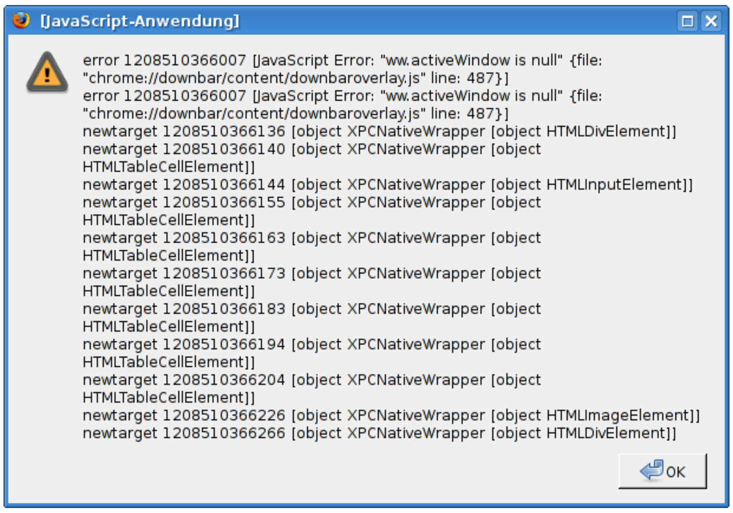
Figure 7.13: A typical dialog between a user and the hotline after an error occurs. The dialog itself is made up, but based on many similar experiences.

- Before sending error reports to the manufacturer, the user should be asked if he agrees with it. This seems to be especially necessary in the context of the Internet, were many users seem to have bad feelings about data being sent to another entity.
- Providing a feedback form can provide a way of active listening. As explained in Chapter 5.3.2), this can be an effective way to reduce the frustration of the user.

7.4.4 Implementation

The Firefox-extension was written as a XUL-based JavaScript. Mozilla's interface allows extensions to register new event listeners and to modify Firefox's (or any other Mozilla-based application) interface by adding new components based on the markup-language XUL. As extensions are running in another security domain, their execution is not affected by JavaScript-errors in the current document (the HTML-file). They therefore also have only restricted access to the document.

This extension adds only one component to the user interface: an entry "Show error analyzer" in the "Tools"-menu, which prints some debug information about the extension's internal memory as can be seen in Figure 7.14.



```
error 1208510366007 [JavaScript Error: "ww.activeWindow is null" {file:
"chrome://downbar/content/downbaroverlay.js" line: 487}]
error 1208510366007 [JavaScript Error: "ww.activeWindow is null" {file:
"chrome://downbar/content/downbaroverlay.js" line: 487}]
newtarget 1208510366136 [object XPCNativeWrapper [object HTMLDivElement]]
newtarget 1208510366140 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366144 [object XPCNativeWrapper [object HTMLInputElement]]
newtarget 1208510366155 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366163 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366173 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366183 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366194 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366204 [object XPCNativeWrapper [object
HTMLTableCellElement]]
newtarget 1208510366226 [object XPCNativeWrapper [object HTMLImageElement]]
newtarget 1208510366266 [object XPCNativeWrapper [object HTMLDivElement]]
```

Figure 7.14: The internal event log of the ErrorAnalyzer. Here, an advertisement banner triggered an error; this is usually not a malfunction that needs to be handled.

The main work for error detection is done by three listeners:

- `onMouseMove`: every time the user moves the mouse, the extension checks if the cursor was moved over a new element in the current HTML-file.
- `onClick`: every click on an element in the HTML-file is recorded.

- `JSErrorAnalyzer_cb` is a listener-object being attached to the console-service in order to receive error messages. The basic principle of registering such a listener is shown in Fig. 7.15.

```
var JSErrorAnalyzer_cb = {
  observe: function( aMessage ){
    if (!aMessage.message.match(/JavaScript Error/))
       return; // No Notices
    // do something
  }
};
var JSErrorAnalyzer = {
  onLoad: function() {
    var aConsoleService =
           Components.classes["@mozilla.org/consoleservice;1"].
           getService(Components.interfaces.nsIConsoleService);
    aConsoleService.registerListener(JSErrorAnalyzer_cb);
  }
}
window.addEventListener("load", JSErrorAnalyzer.onLoad, false);
```

Figure 7.15: Registering an error handler on a website.

The extension keeps an internal history of events which is filled by the event listeners (and cleared on a regular base and after loading a new page, to prevent memory leaks). Once an error is thrown, the program checks if one of the following two conditions are met:

- The user has clicked on an element within the last 500ms.
- The user moved the cursor over a new element within the last 500ms and has moved the cursor above the same element before within the last 3s.

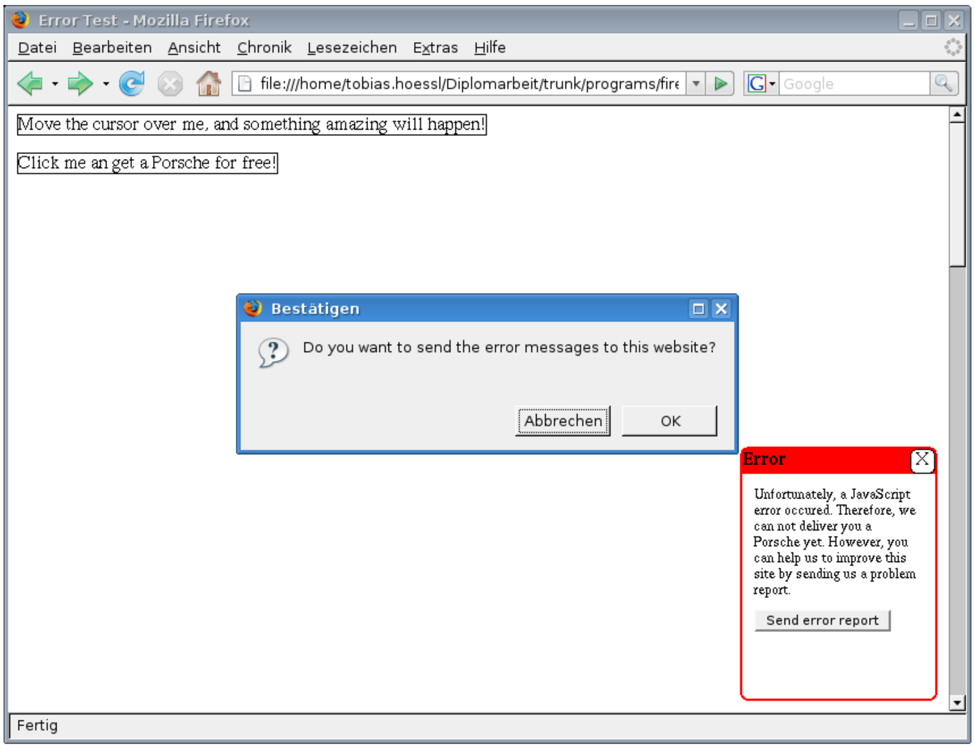
If one of these conditions is met, a malfunction is assumed and error handling starts. A alert box like shown in Figure 7.17 is displayed at the bottom right of the browser window. The content of this window can be either:

- A website-dependent HTML-file, embedded into an IFRAME. The location of this file can be specified by the webmaster by including a meta-tag into the HEAD-part of the HTML-file (as shown in Fig. 7.16).
- A default error message, informing the user that a JavaScript error has occurred, if no website-dependent error handler is defined. No report to the webmaster is sent in that case.

```
<HEAD>
  <TITLE>Error Test</TITLE>
  <META name="errorlink" content=
    "http://www.hoessl.org/diplomarbeit/error-backend-iframe.php" />
</HEAD>
```

Figure 7.16: Registering an error handler on a website.

Again, the graphical design of the error wizard is not within the scope of this thesis, but rather the underlying mechanisms.


Error Test - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
file:///home/tobias.hoessl/Diplomarbeit/trunk/programs/fire
Google
Bestätigen
Abbrechen
OK
Error
Send error report
Fertig

Move the cursor over me, and something amazing will happen!

Click me an get a Porsche for free!

Do you want to send the error messages to this website?

Unfortunately, a JavaScript error occured. Therefore, we can not deliver you a Porsche yet. However, you can help us to improve this site by sending us a problem report.

Figure 7.17: Top left: the elements triggering the error. Bottom right: the error frame. Middle: a dialog triggered by the error frame, asking the user to send the internal data.

Once the error window appears, the user can be asked to send a problem report. The problem report includes the error message thrown in the error console, and potentially more information about the system configuration[35].

However, for privacy reasons, normal JavaScripts running in the security domain of websites cannot access the error console, neither can they directly call functions of extensions. The current implementation of the extension therefore uses an event-based approach:

1. The Firefox-extension registers an event listener on the custom event “ErrorInfoRequestEvent”.

2. If the JavaScript of the error frame (coming from the untrusted website-manufacturer domain) wants access to the error messages, it triggers this event. This can be done automatically or after the user clicks a button like “Send a problem report”.

3. Once the extension receives the event, a confirmation from the user is requested.

4. If the user agrees, the error messages are put into the “value”-attribute of the event’s target element, which could be, for example, a hidden input field. Another attribute is set, indicating the success of the operation is set.

5. The JavaScript of the error frame waits until the attributes of step 4 are set. After this, it goes on to send the error message to the manufacturer.

Further interaction, like adding a text field to allow free text and other feedback by the user can be included into the site which the error frame links to.

7.4.5 Limitations of this approach

The main limitation of this concrete implementation is the limited amount of information the extension receives through the API. Three pieces of information can be received:

[35]The browser identification does not have to be sent explicitly, as it is included in every standard HTTP-request.

- The error message (like “Error: win_a_porsche is not defined”).
- The source file where the error occurred.
- And the line number of the source file.

While this already can be quite helpful for the manufacturer trying to figure out the reason for the malfunction, more detailed information like a stack trace could be even more helpful. However, to get this kind of information, this possibility would have to be included into the actual JavaScript-engine of the browser.

The accuracy of the error detection could be improved by having more information about the elements the user is currently interacting with: if occurring errors only were to be correlated with `click`- and `mouseover`-events of elements actually having JavaScript-code attached to them, the number of false alarms could be further reduced.

However, figuring out if any code is attached to an element seems to be a bit tricky, as there are several different ways to attach code:

- Using the DOM-method `addEventListener`.
- Using HTML-attributes (like `onClick=”code`”).
- Using JavaScript-code encoded as an URI (like `href=”javascript:alert(’test’);`”).

7.4.6 Evaluating it

Unfortunately, due to time constraints of this thesis, the prototype of this system could not be completely finalized and evaluated. However, from personal experiences while having the extension installed while surfing, the hit rate seems to be quite satisfying - only with some strange race conditions when multiple tabs are involved.

The error reporting system is, of cause, not used by and real website.

To judge on the real benefit of the system, several questions have to be evaluated:

- Is the recognition rate good enough to cover common cases of malfunctions while avoiding false alarms?
- Does a well-designed error dialog help to reduce user frustration?
- How frequently do users use the feedback function, compared to normal feedback forms, in the case of a malfunction? Is the gained information helpful for the manufacturer in order to figure out the problem?

7.5 Discussion of the results

The results of the two evaluated systems indeed suggest, that user behavior can be used as a valuable source of information for malfunction recognition and response.

In the second iteration of the Microphone Helper, the system recognized all attempts by the participants to use it - the problematic part seemed to be rather the user interaction of the help wizard than the actual detection and recovery. Also, a long-term study might be needed to find out how many false alarms the system produces in real world scenarios.

The problems of the Microphone Helper’s first iteration indicate the importance to carefully design the user interaction with the special situation of the malfunction in mind: the user might be fixated on something specific at that moment and therefore is less likely to notice help offers by the computer. If the malfunction is really frustrating for the user, psychological effects mentioned in Chapter 5.2 (regression, fixation, and so on) might further affect the user interaction. Designing help systems that are flashy enough to be noticed in such a situation, yet as little annoying as possible in a situation of a false alarm, is probably quite a challenge.

The results of the second iteration clearly show the potential of such a system: all participants could solve the problem, with an average time of 21 seconds; this is much better than the control group, who had to solve the rather difficult problem themselves (and mostly did not succeed at all). However, compared to very easy problems that users can solve immediately themselves, it appears that this system is a bit slower, although not significantly. This suggests that behavior-based malfunction response has the biggest benefit for complex problems and for less experienced users.

The results of the mouse helper impressively show how reliable error symptoms can trigger a specific user behavior - namely the rapid mouse movements as a result of a frozen mouse cursor: this behavior could be successful recognized in 87% of the tests. According to the data collected during normal office work, the rate of false alarms is low (only two false alerts in 30 hours of work) and can further reduced by additional tracking.

The results of the survey for the Microphone Helper suggest that there are two groups of users: users preferring to solve such problem themselves, and users who prefer to let the computer do this. The first group was bigger in this survey, which might be due to the relatively high computer experience of the participants. When designing actual systems, this fact should be considered, to let the user choose whether to use automatic problem recovery or not.

The users generally liked the idea of the computer actively responding to malfunctions. On the 5-point Likert scale, the answers given to this statement was nearly identical in all groups: 3.47 for users of the Microphone helper, 3.50 for the control group, and again 3.47 for the users of the Mouse helper.

The results do not suggest that the participants of the surveys were overly concerned about the privacy issues of the computer monitoring their behavior: 2.38 on the item “I'm bothered by the thought that the computer would always listen, even if the microphone is muted” in the case of the microphone helper, and 2.53 in the analogous question for the Mouse helper. Personally, I doubt this low value is really representative. Another study with more participants and more specific questions should be conducted to confirm or vitiate these findings.

8 Future work

This chapter points out work that can be done in the future to bring forward the research about behavior-based malfunction response.

8.1 Problem sources

It appears that the exact sources of frustration with computers are changing over time; Lazar et al. [13] found in 2003 [36], that “system crashes were the most commonly-reported frustrating experience”. From the results of the surveys in this thesis, it seems that now, half a decade later, complete system crashes are by far not as common anymore, and the main source of frustration has shifted to other components. However, the research done in this thesis about problem sources in the real world is by far not as systematic as Lazar’s methodology. Therefore, more research, with more participants and a quantitative approach, would be of great help for this field of research.

The same goes for the reactions of the users - not only the physical actions, but also for the psychological effects, as they might greatly affect the further interaction.

8.2 Malfunction framework

The explanations about the Mouse Helper’s implementation pointed out the need for a protocol to exchange information about malfunctions and user behavior between different layers in the system. This does not only affect the interface between the hardware and operating system layer, but also the interface between the operating system and the application layer, and, maybe a bit less important, the interface between the application and the document layer.

This raises two major questions, up to further research:

- How can information about malfunctions be formalized, and
- How well can such information be embedded into existing interfaces and protocols (for example, as in the Mouse Helper case, into the USB-protocol)?

8.3 Interface design

The experience made with the Microphone Helper shows that special attention has to be put onto the user’s situation when responding on malfunctions. Therefore, developing more specific guidelines about how to design this special kind of user feedback would be helpful, to manage the balancing act of being flashy enough without being too obtrusive.

8.4 Practical projects

8.4.1 Frustration 2.0

The Firefox-extension was developed in principle, but has not been widely tested in real-world scenarios and still has several bugs. Therefore, it might be worth it to polish it further, add a proper user interface and evaluate its hit ratio.

It also would be interesting to analyze if an active listening agent like in “This computer responds to user frustration” [11] can help to reduce frustration in this setting as well. The agent would have to be implemented in the domain of the manufacturer (not in the domain of the Firefox extension), using a server-side language like PHP.

[36] The paper was published 2006, however the study was also presented on America’s Conference on Information Systems (AMCIS) 2003.

8.4.2 Malfunction detection using a microphone

Deeper analysis of the sound input of a microphone might be used to reveal more kinds of malfunctions. Some ideas besides the usage of the Microphone Helper are:

- The computer could recognize malfunctions in the sound output. If the sound card produces output and a sound cable is plugged in, but no sound output can be heard using the microphone, then there might be a problem with the external amplifier or the loudspeakers. The computer can probably not do much about it, but help the user to find the problem.
- Typical annoying noises can maybe be recognized. In the web survey, sound interferences when receiving a text message on the mobile phone and beeping CRT monitors were mentioned as hardware problems. The computer could give hints how to prevent these noises as a response.
- The computer could notice curses of the user and take this as an indicator that something is possibly not going well. However, this overlaps much with the work already done by Schuller [27].

Analyzing the possibilities of such a microphone-based system should also emphasis the privacy aspects of monitoring the user to such an extent more in detail.

9 Conclusion

System malfunction recognition and response based on user behavior is a field of research that is still widely unexplored, but promises new perspectives on error handling. This thesis contributes to the research in two ways: On the one hand, a model and taxonomies for behavior-based malfunction response were proposed. While this model is not perfect yet, it provides an initial systematization. One the other hand, the practical projects of this thesis demonstrated that user behavior can not only be used for malfunction detection in theory, but in real-world scenarios as well.

A lot of work still has to be done in researching behavior-based malfunction response. Neither do we know the full potential of it yet, nor do we know about all peculiarities and implications of such systems. But the findings of this thesis suggest that is is indeed worth to look further into behavior-based malfunction response.

References

[1] R. Barker and T. Dembo. Frustration and regression: An experiment with young children. In *Frustration: The Development of a Scientific Concept*. MacMillan Publishing Co., 1965.

[2] Patrick Baudisch, John Pruitt, and Steve Ball. Flat volume control: Improving usability by hiding the volume control hierarchy in the user interface. Technical report, Microsoft Research / MSX / eHome, 2004.

[3] Katie Bessiere, Irina Ceaparu, Jonathan Lazar, John Robinson, and Ben Shneiderman. *Social and Psychological Influences on Computer User Frustration*, pages 169–192. Lawrence Erlbaum Associates, 2004.

[4] Marleen Brinks. *Aggression gegen Computer - Eine wissenschaftliche Untersuchung eines alltäglichen Phänomens.* ibidem-Verlag Stuttgart, 2005.

[5] S. H. Britt and S. Q. Janus. Criteria of frustration. *The Psychological Review*, 47(6):451–469, 1940.

[6] Irina Ceaparu, Jonathan Lazar, Katie Bessiere, John Robinson, and Ben Shneiderman. Determining causes and severity of end-user frustration, May 2002.

[7] J. Dollard, L. W. Doob, N. E. Miller, O. H. Mowrer, and R. R. Sears. *Frustration and Aggression.* Yale University Press, New Haven, 1939.

[8] Peter Hyland and Lejla Vrazalic. Applying a taxonomy of error in usability testing. Technical report, University of Wollongong, 2003.

[9] International Ergonomics Association. *Frustrating Computer Users Increases Exposure to Physical Factors*, 2003.

[10] Francis Jambon. Error recovery representations in interactive system development. In *Third Annual ERCIM Workshop on "User Interfaces for All"*, pages 177–182, 1997.

[11] Jonathan Klein, Youngme Moon, and Rosalind W. Picard. This computer responds to user frustration: Theory, design, results. *Interacting with Computers*, 14(2):119–140, 2002.

[12] Tom Kontogiannis. User strategies in recovering from errors in man-machine systems. *Safety Science*, 32:49–68, 1999.

[13] Jonathan Lazar, Adam Jones, Katie Bessiere, Irina Ceaparu, and Ben Shneiderman. Workplace user frustration with computers: An exploratory investigation of the causes and severity. *Behaviour & Information Technology*, 25(3), 2006.

[14] Jonathan Lazar, Adam Jones, and Mary Hackley. Severity and impact of computer user frustration: A comparison of student and workplace users. *Interacting with Computers*, 18(2):187–207, 2002.

[15] Clayton Lewis and Donald A. Norman. Designing for error. In *User Centered System Design: New Perspectives on Human-Computer Interaction*, pages 411–432. Lawrence Erlbaum Associates, 1986.

[16] Steven W. Macbeth et al. Monitoring group activities (patent 20070300174), 2006.

[17] Helena M. Mentis and Geri K. Gay. Using touchpad pressure to detect negative affect. In *Fourth IEEE International Conference on Multimodal Interfaces*, pages 406–410. IEEE Computer Society, 2002.

[18] Jin Mo and Yves Crouzet. Human error tolerant design for air-traffic control systems. In *Proceedings of the Third Probabilistic Safety Assessment and Management Conference*, pages 400–405, 1996.

[19] MORI. Employees get 'it' out of their systems, 1999. [Website offline by now; snapshot included into the DVD].

[20] Hans-Peter Nolting. *Lernfall Aggression.* Rohwolt Taschenbuch Verlag, Reinbek, 2005.

[21] Kent L. Norman. Computer rage: Exploring distributions of frustration and dealing with issues in the analysis of an online survey. Technical report, University of Maryland, 2006.

[22] Rosalind W. Picard. Towards computers that recognize and respond to user emotion. *IBM Systems Journal*, 39(3):705–719, 2000.

[23] Carson Jonathan Reynolds. The sensing and measurement of frustration with computers. Master's thesis, Massachusetts Institute of Technology, 2001.

[24] Carson Jonathan Reynolds and Rosalind W. Picard. Affective sensors, privacy, and ethical contracts. In *CHI '04 extended abstracts on Human factors in computing systems*, 2004.

[25] S. Rosenzweig. Tests of frustration. *American Journal of Orthopsychiatry*, 5:395–403, 1935.

[26] Jocelyn Scheirer, Raul Fernandez, Jonathan Klein, and Rosalind W. Picard. Frustrating the user on purpose: a step toward building an affective computer. *Interacting with Computers*, 14(2):93–118, 2002.

[27] Björn Schuller. *Automatische Emotionserkennung aus sprachlicher und manueller Interaktion.* PhD thesis, Technische Universität München, 2006.

[28] Michael W. Shapiro. Self-healing in modern operating systems. *ACM Queue*, 2(8), November 2004.

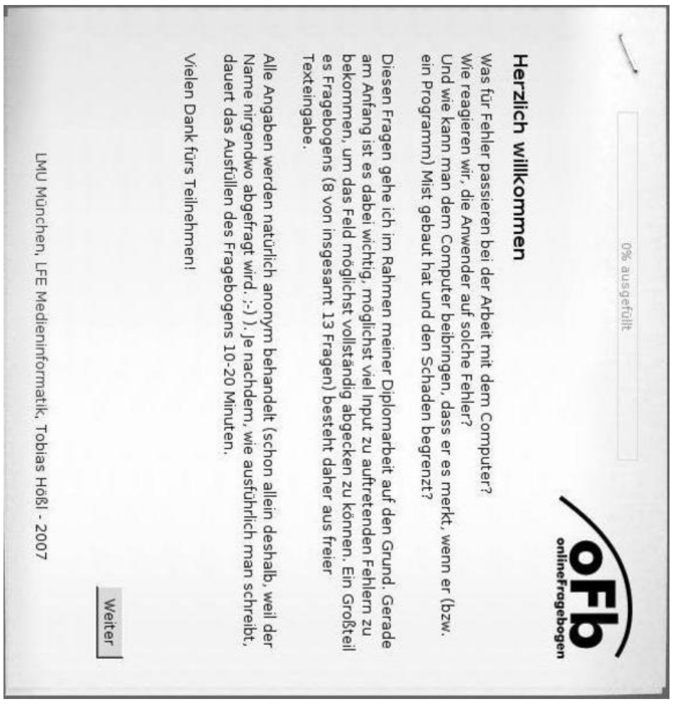
[29] Burrhus Frederic Skinner. 'superstition' in the pigeon. *Journal of Experimental Psychology*, 38:168–172, 1948.

[30] D.D. Woods. Some results on operator performance in emergency events. In *Institute of Chemical Engineers Symposium Series*, volume 90, pages 21–31, 1984.

[31] D. Zapf, F.C. Brodbeck, M. Frese, H. Peters, and J. Prümper. Errors in working with office computers: A first validation of a taxonomy for observed errors in a field setting. *International Journal of Human-Computer Interaction*, 4:311–339, 1992.

A Surveys

A.1 Qualitative web-survey



0% ausgefüllt

oFb onlineFragebogen

Herzlich willkommen

Was für Fehler passieren bei der Arbeit mit dem Computer?
Wie reagieren wir, die Anwender auf solche Fehler?
Und wie kann man dem Computer beibringen, dass er es merkt, wenn er (bzw. ein Programm) Mist gebaut hat und den Schaden begrenzt?

Diesen Fragen gehe ich im Rahmen meiner Diplomarbeit auf den Grund. Gerade am Anfang ist es dabei wichtig, möglichst viel Input zu auftretenden Fehlern zu bekommen, um das Feld möglichst vollständig abgecken zu können. Ein Großteil es Fragebogens (8 von insgesamt 13 Fragen) besteht daher aus freier Texteingabe.

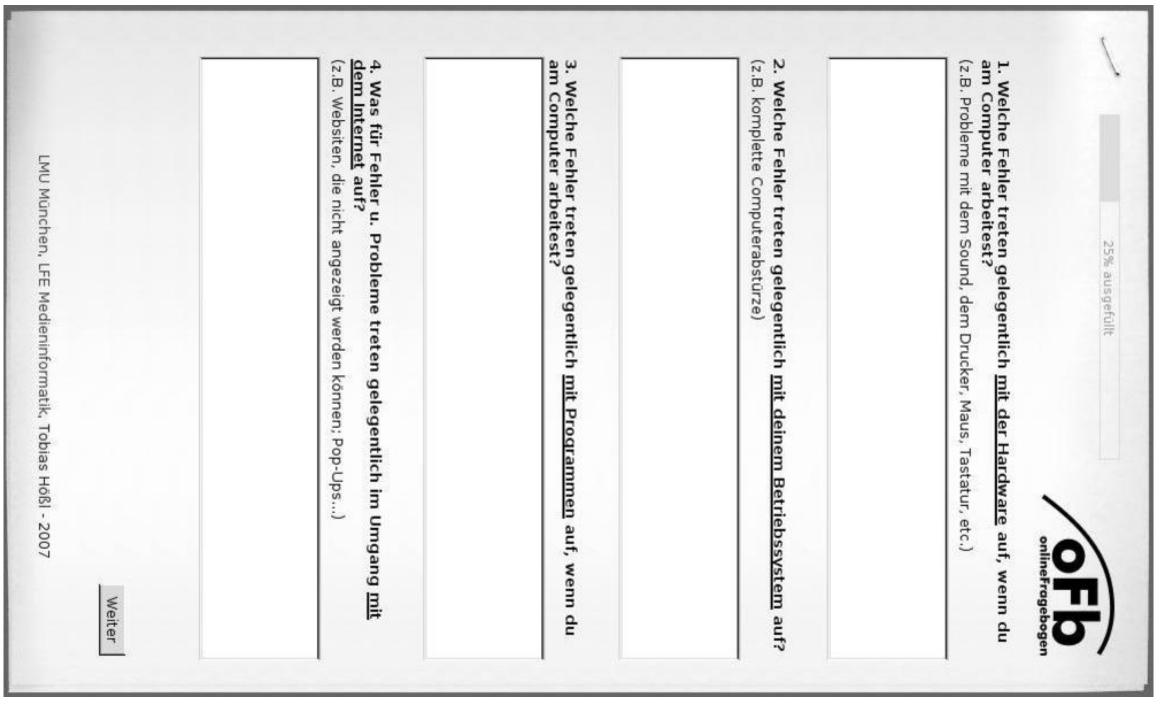
Alle Angaben werden natürlich anonym behandelt (schon allein deshalb, weil der Name nirgendwo abgefragt wird. ;-)). Je nachdem, wie ausführlich man schreibt, dauert das Ausfüllen des Fragebogens 10-20 Minuten.

Vielen Dank fürs Teilnehmen!

Weiter

LMU München, LFE Medieninformatik, Tobias Hößl - 2007

Figure A.1: Page 1 of the web-survey.



25% ausgefüllt

oFb onlineFragebogen

1. Welche Fehler treten gelegentlich mit der Hardware auf, wenn du am Computer arbeitest?
(z.B. Probleme mit dem Sound, dem Drucker, Maus, Tastatur, etc.)

2. Welche Fehler treten gelegentlich mit deinem Betriebssystem auf?
(z.B. komplette Computerabstürze)

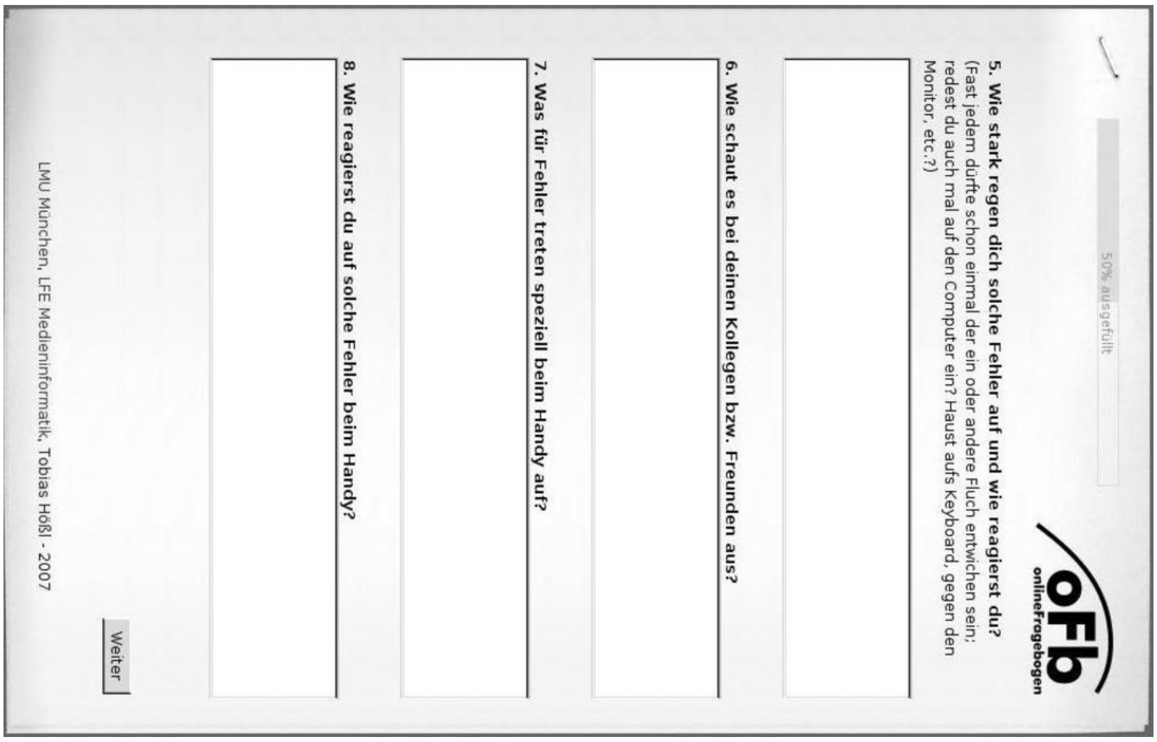
3. Welche Fehler treten gelegentlich mit Programmen auf, wenn du am Computer arbeitest?

4. Was für Fehler u. Probleme treten gelegentlich im Umgang mit dem Internet auf?
(z.B. Websiten, die nicht angezeigt werden können; Pop-Ups...)

Weiter

LMU München, LFE Medieninformatik, Tobias Hößl - 2007

Figure A.2: Page 2 of the web-survey.



50% ausgefüllt

onlineFragebogen

5. Wie stark regen dich solche Fehler auf und wie reagierst du?
(Fast jedem dürfte schon einmal der ein oder andere Fluch entwichen sein; redest du auch mal auf den Computer ein? Haust aufs Keyboard, gegen den Monitor, etc.?)

6. Wie schaut es bei deinen Kollegen bzw. Freunden aus?

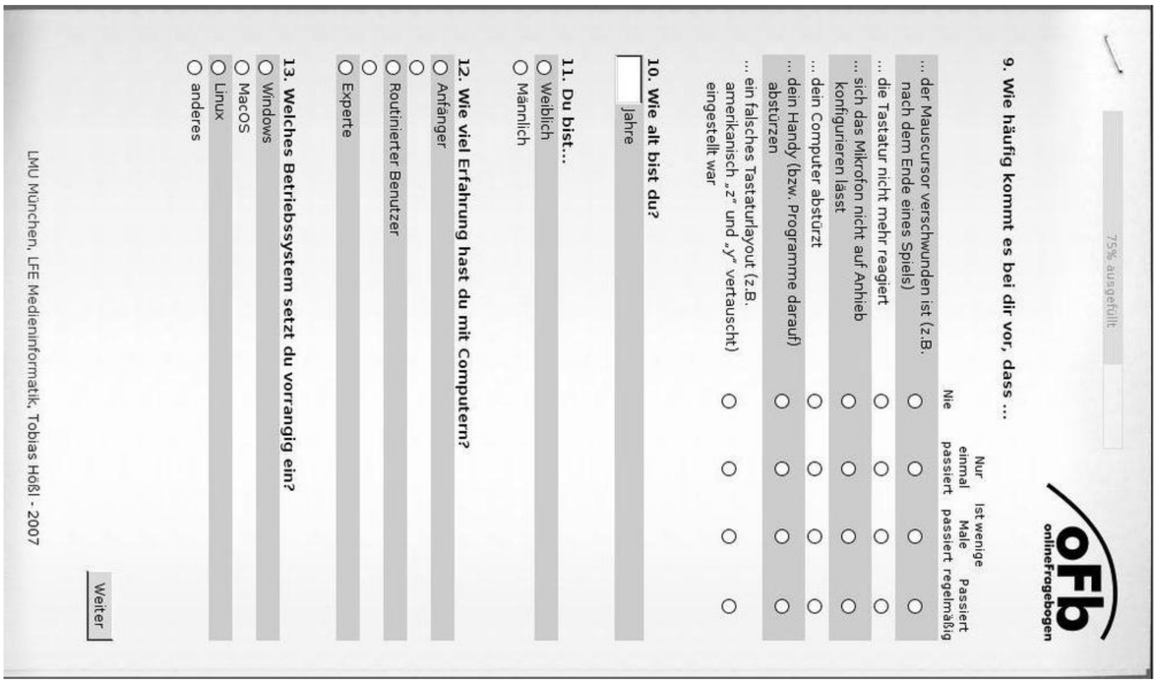
7. Was für Fehler treten speziell beim Handy auf?

8. Wie reagierst du auf solche Fehler beim Handy?

Weiter

LMU München, LFE Medieninformatik, Tobias Hößl - 2007

Figure A.3: Page 3 of the web-survey.



75% ausgefüllt

onlineFragebogen

9. Wie häufig kommt es bei dir vor, dass ...

	Nie	Nur einmal passiert	Ist wenige Male passiert	Passiert regelmäßig
... der Mauscursor verschwunden ist (z.B. nach dem Ende eines Spiels)	○	○	○	○
... die Tastatur nicht mehr reagiert	○	○	○	○
... sich das Mikrofon nicht auf Anhieb konfigurieren lässt	○	○	○	○
... dein Computer abstürzt	○	○	○	○
... dein Handy (bzw. Programme darauf) abstürzen	○	○	○	○
... ein falsches Tastaturlayout (z.B. amerikanisch, „z" und „y" vertauscht) eingestellt war	○	○	○	○

10. Wie alt bist du?
[ ] Jahre

11. Du bist...
○ Weiblich
○ Männlich

12. Wie viel Erfahrung hast du mit Computern?
○ Anfänger
○
○ Routinierter Benutzer
○
○ Experte

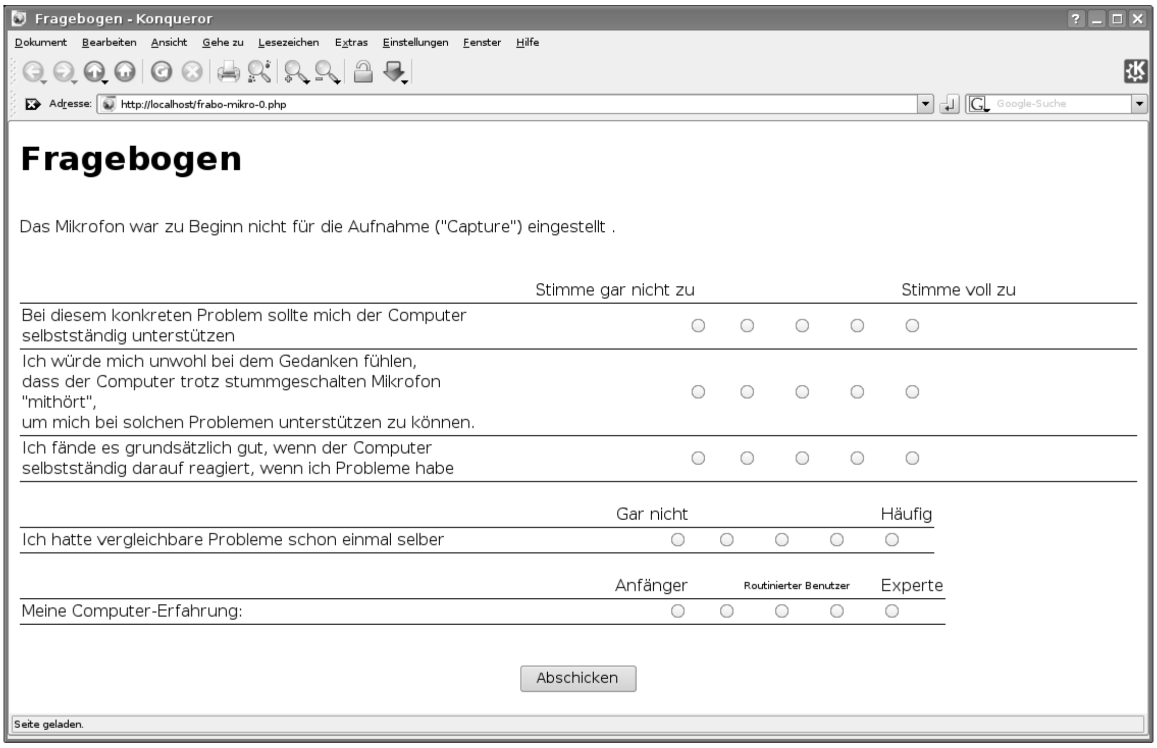
13. Welches Betriebssystem setzt du vorrangig ein?
○ Windows
○ MacOS
○ Linux
○ anderes

Weiter

LMU München, LFE Medieninformatik, Tobias Hößl - 2007

Figure A.4: Page 4 of the web-survey.

A.2 Survey of the Microphone helper


Fragebogen - Konqueror
Dokument Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe
Adresse: http://localhost/frabo-mikro-0.php
Google-Suche
Seite geladen.

Fragebogen

Das Mikrofon war zu Beginn nicht für die Aufnahme ("Capture") eingestellt .

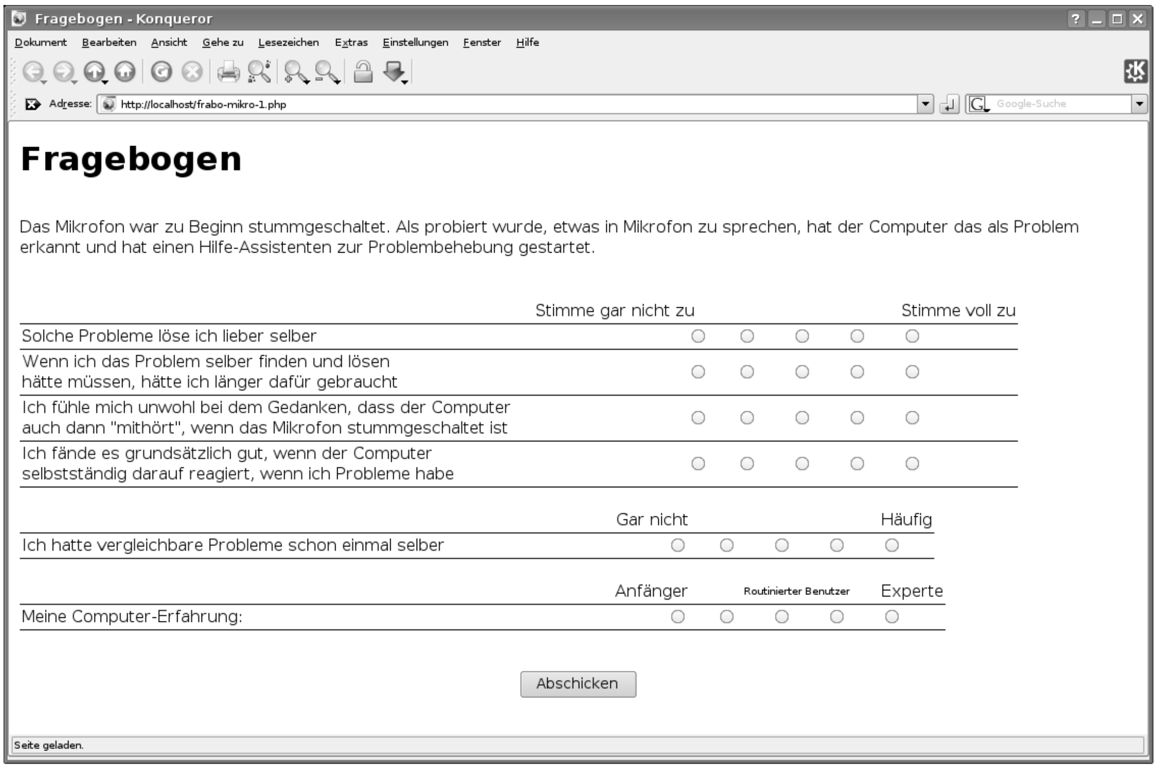
	Stimme gar nicht zu				Stimme voll zu
Bei diesem konkreten Problem sollte mich der Computer selbstständig unterstützen	○	○	○	○	○
Ich würde mich unwohl bei dem Gedanken fühlen, dass der Computer trotz stummgeschalten Mikrofon "mithört", um mich bei solchen Problemen unterstützen zu können.	○	○	○	○	○
Ich fände es grundsätzlich gut, wenn der Computer selbstständig darauf reagiert, wenn ich Probleme habe	○	○	○	○	○

	Gar nicht				Häufig
Ich hatte vergleichbare Probleme schon einmal selber	○	○	○	○	○

	Anfänger		Routinierter Benutzer		Experte
Meine Computer-Erfahrung:	○	○	○	○	○

Abschicken

Figure A.5: The survey for the control group.


Fragebogen - Konqueror
Dokument Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Fenster Hilfe
Adresse: http://localhost/frabo-mikro-1.php
Google-Suche
Seite geladen.

Fragebogen

Das Mikrofon war zu Beginn stummgeschaltet. Als probiert wurde, etwas in Mikrofon zu sprechen, hat der Computer das als Problem erkannt und hat einen Hilfe-Assistenten zur Problembehebung gestartet.

	Stimme gar nicht zu				Stimme voll zu
Solche Probleme löse ich lieber selber	○	○	○	○	○
Wenn ich das Problem selber finden und lösen hätte müssen, hätte ich länger dafür gebraucht	○	○	○	○	○
Ich fühle mich unwohl bei dem Gedanken, dass der Computer auch dann "mithört", wenn das Mikrofon stummgeschaltet ist	○	○	○	○	○
Ich fände es grundsätzlich gut, wenn der Computer selbstständig darauf reagiert, wenn ich Probleme habe	○	○	○	○	○

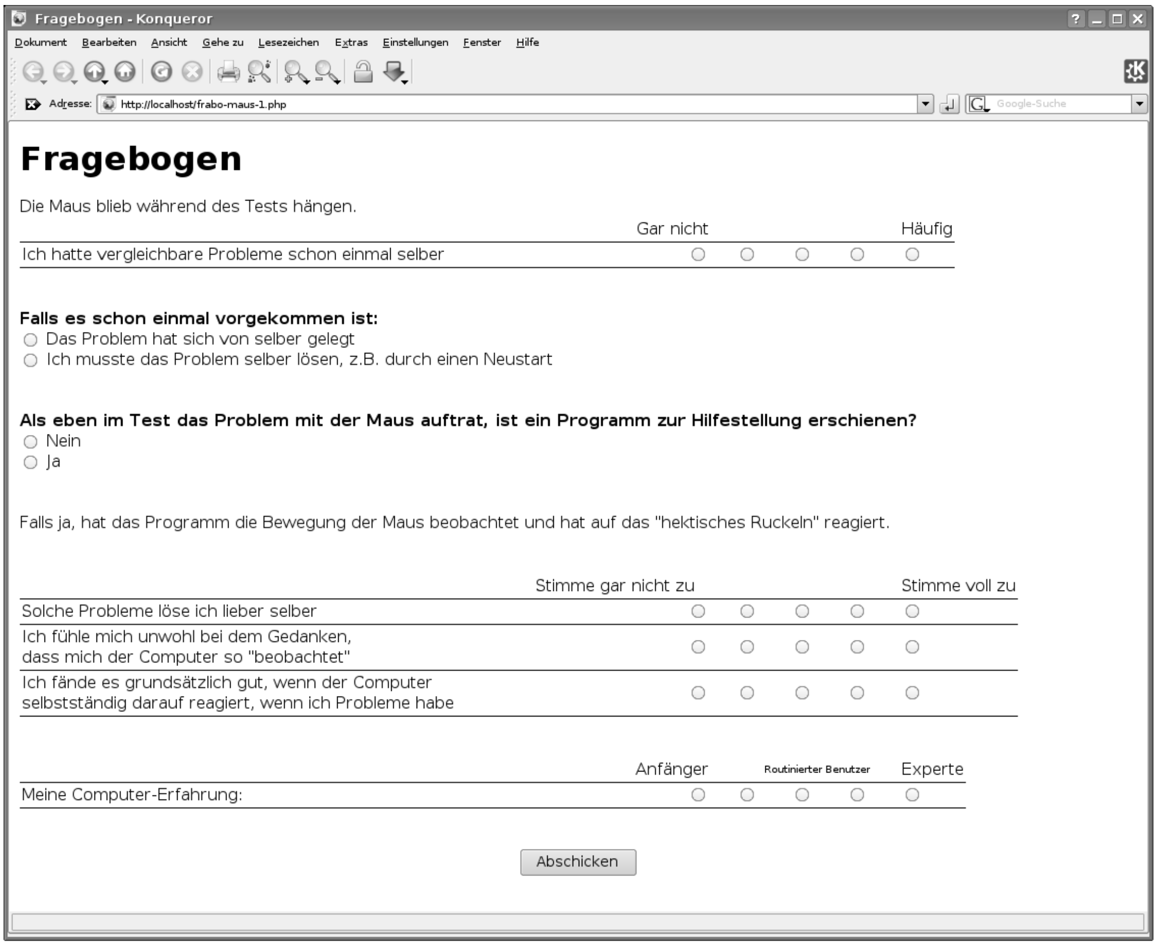
	Gar nicht				Häufig
Ich hatte vergleichbare Probleme schon einmal selber	○	○	○	○	○

	Anfänger		Routinierter Benutzer		Experte
Meine Computer-Erfahrung:	○	○	○	○	○

Abschicken

Figure A.6: The survey for the users of the Microphone Helper.

A.3 Survey of the Mouse helper


Fragebogen - Konqueror

Adresse: http://localhost/frabo-maus-1.php

Fragebogen

Die Maus blieb während des Tests hängen.

	Gar nicht				Häufig
Ich hatte vergleichbare Probleme schon einmal selber	○	○	○	○	○

Falls es schon einmal vorgekommen ist:
○ Das Problem hat sich von selber gelegt
○ Ich musste das Problem selber lösen, z.B. durch einen Neustart

Als eben im Test das Problem mit der Maus auftrat, ist ein Programm zur Hilfestellung erschienen?
○ Nein
○ Ja

Falls ja, hat das Programm die Bewegung der Maus beobachtet und hat auf das "hektisches Ruckeln" reagiert.

	Stimme gar nicht zu				Stimme voll zu
Solche Probleme löse ich lieber selber	○	○	○	○	○
Ich fühle mich unwohl bei dem Gedanken, dass mich der Computer so "beobachtet"	○	○	○	○	○
Ich fände es grundsätzlich gut, wenn der Computer selbstständig darauf reagiert, wenn ich Probleme habe	○	○	○	○	○

	Anfänger		Routinierter Benutzer		Experte
Meine Computer-Erfahrung:	○	○	○	○	○

Abschicken

Figure A.7: The survey of the Mouse Helper study.