User Interface Defect Detection by Hesitation Analysis

Robert W. Reeder and Roy A. Maxion
reeder@cs.cmu.edu, maxion@cs.cmu.edu

Dependable Systems Laboratory
Computer Science Department
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213 / USA

Abstract

Delays and errors are the frequent consequences of people having difficulty with a user interface. Such delays and errors can result in severe problems, particularly for mission-critical applications in which speed and accuracy are of the essence. User difficulty is often caused by interface-design defects that confuse or mislead users. Current techniques for isolating such defects are time-consuming and expensive, because they require human analysts to identify the points at which users experience difficulty; only then can diagnosis and repair of the defects take place. This paper presents an automated method for detecting instances of user difficulty based on identifying hesitations during system use. The method's accuracy was evaluated by comparing its judgments of user difficulty with ground truth generated by human analysts. The method's accuracy at a range of threshold parameter values is given; representative points include 92% of periods of user difficulty identified (with a 35% false-alarm rate); 86% (24% false-alarm rate); and 20% (3% false-alarm rate). Applications of the method to addressing interface defects are discussed.

1 Introduction

Undependable user interfaces present a major obstacle to achieving overall system dependability. Avizienis et al. [1], in outlining current challenges to dependable computing, state this in no uncertain terms:

> The problems of complex human-machine interactions (including user interfaces) remain a challenge that is becoming very critical – the means to improve their dependability and security need to be identified and incorporated.

User interface dependability can be defined as the extent to which a user interface allows human users to complete intended goals within specified speed and accuracy targets. When an interface prevents users from meeting these speed and accuracy targets, defects in the design of the interface are likely to be at fault. Following Maxion and deChambeau [16], a user interface defect is defined here as any aspect of a user interface that impairs a user's ability to achieve a goal. Examples of such defects include ambiguous labels on interface controls, incomprehensible instructions, confusing icons, and inadequate system-status feedback. Such defects can confuse users, causing delays and errors in users' progress toward goal completion.

Interface dependability can only be achieved if interface defects are detected, diagnosed, and recovered from. However, merely detecting user-interface defects has proven to be a very difficult problem. A variety of techniques, including inspection-based methods (usability experts inspect interface designs for defects), user modeling (a software model of a human user is built and used to simulate interaction with an interface), user opinion surveys, field observation of users at work, and laboratory user testing have been studied by researchers and used in practice [2, 5, 11]. However, all of these methods have significant weaknesses, including both failure to detect defects (misses) and classification of non-problematic aspects of interfaces as defects (false alarms). Of all available methods, observation-based techniques, i.e., field observation and laboratory user testing, are generally accepted as the best for acquiring valid results. Nielsen [19, p. 165] says:

> [T]esting with real users is the most fundamental usability method and is in some sense irreplaceable, since it provides direct information about how people use computers and what their exact problems are with the concrete interface being tested.

Proceedings of the 2006 International Conference on Dependable Systems and Networks (DSN'06)
0-7695-2607-1/06 $20.00 © 2006 IEEE

The primary drawbacks of observation-based techniques are their expense in terms of usability-analyst time, and their unreliability due to analyst error. Whether conducted in the field or in the laboratory, these techniques require one or more expert usability analysts to observe user actions with an interface, and to record instances of user difficulty, which is symptomatic of an interface defect. Usability-analyst time can be quite expensive; Nielsen valued it at US$100 per hour in 1993 [21]. The expense of usability-analyst time often limits the amount of user data that can be covered. Thus, rarely-manifested defects may be missed during testing. Besides their expense, usability analysts are error prone; they're human. They may fail to notice instances of user difficulty, either because such instances occur too rapidly for the analyst to record them all, or because they occur so infrequently that analysts lose vigilance [19, 23]. Analyst error is another cause of missed defects. Although this kind of error can be mitigated by videotaping and later reviewing user sessions, reviewing videotapes introduces more expense; Nielsen estimates that reviewing videotape can take three to ten times as long as the original user test [19].

This paper introduces a method, called hesitation detection, or automatically detecting instances of user difficulty. An instance of user difficulty is defined as an instance in which a user's ability to achieve a goal is impaired; so, by definition, user difficulty is symptomatic of an interface defect. The method identifies hesitations, defined as anomalously long pauses, in users' interactions with the mouse and keyboard. Although such hesitations can occur for many reasons, they often indicate user difficulty. Like other methods for interface defect detection, hesitation detection accuracy can be characterized by two measures: first, by the percentage of all user difficulties that it detects (the hit rate), and second, by the percentage of benign events that it mistakenly classifies as difficulties (the false-alarm rate). This paper addresses the question of how accurate, in terms of hit rate and false alarm rate, hesitation detection is at detecting instances of user difficulty and, hence, interface defects.

Assuming it does have the ability to detect instances of user difficulty accurately, hesitation detection provides several significant enhancements to observation by a human usability analyst alone. First, it is cheap; since hesitation detection is automated, it can save human-analyst time. Second, it provides better coverage; much more data can be searched for instances of user difficulty than could feasibly be searched by a human analyst alone. Third, it is immune to human error; it does not miss instances of user difficulty due to limited attention or to lack of vigilance. Finally, its results are repeatable; it is a deterministic method.

To address the question of hesitation detection accuracy, a hesitation detector was implemented and its output was compared with a usability analyst's ratings of user difficulty for a laboratory user study. In the user study, usability data were collected from users performing tasks with two different interfaces for setting file permissions on a Windows XP system. The users were instructed to think aloud as they worked, explaining what they were doing at each point in the task [6]. Mouse and keyboard data were collected and subjected to hesitation analysis, while screen video and think-aloud audio recordings were collected and subjected to analysis by a human usability expert. Both methods, hesitation detection and expert analysis, produced judgments of periods during which users had difficulty with the interfaces.

Using expert judgment as ground truth, the hit and false-alarm rates of the hesitation detector were computed. The hesitation-detection method allows these accuracy rates to be tuned according to a sensitivity parameter, so a receiver operating characteristic (ROC) curve, which shows the tradeoff between hit and false-alarm rates as the sensitivity parameter is adjusted, was computed. Representative points on the ROC curve include a 100% hit rate with a 63% false-alarm rate; a 92% hit rate with a 35% false-alarm rate; an 86% hit rate with a 24% false-alarm rate; and a 20% hit rate with a 3% false alarm rate.

Although these results may at first appear disappointing, they are in fact quite good, providing a substantial improvement over previous results, and providing a foundation for huge economies in analyst time savings. For example, at one detector sensitivity value, under very conservative assumptions about how an analyst works, 60% of an analyst's time can be recovered while still detecting 81% of all defects. This computation assumes an ideal analyst who requires only one viewing of the data; for any real analyst, time savings would be as high as 96%. Furthermore, the saved time could be used to scan more data, which would reveal more defects, possibly including those not previously presented to the detector. It is concluded that hesitation detection can provide significant aid to human analysts in detecting interface defects.

2 Objective and approach

This paper addresses the following research question:

> With what accuracy can instances of user difficulty be detected through automated hesitation detection?

This question is important because hesitations do not always indicate difficulty, and user difficulty is not always manifested as hesitations. So, in order to determine whether hesitation detection has any potential application to interface defect detection, it is first necessary to measure its accuracy. To clarify this research question, the terms *hesitation detection*, *user difficulty*, and *accuracy* are defined and explained forthwith.

2.1 Hesitation detection method

The hesitation detector employed in this study is an algorithm that takes as input a time-stamped, chronologically ordered data stream of mouse and keyboard events, including mouse movements, mouse clicks, and keystrokes. It outputs a list of hesitations – anomalously long pauses between events in the data stream. Anomalously long pauses are identified by computing the latency between every pair of consecutive events in the data stream, computing the average-length latency between events, and outputting those latencies that exceed a certain threshold. The threshold is specified as a number of standard deviations from the average-length latency. The threshold value serves as a sensitivity parameter for the detector. A low threshold value will cause the detector to classify more pauses as hesitations, thus potentially outputting more hits at the risk of also outputting more false alarms. A high threshold value will cause the detector to classify fewer pauses as hesitations, thus potentially reducing false alarm output at the risk of more misses. Note that individual differences in mouse and keyboard activity are taken into account by computing a latency average and standard deviation *independently for each user*. Thus, hesitations are defined relative to each user, and are still valid for users who use input devices unusually quickly or slowly.

2.2 User difficulty

User difficulty is, conceptually, an internal cognitive state in which the ability to achieve a goal is impaired. This state can be characterized colloquially as confusion, frustration, uncertainty, lack of knowledge, indecision, etc. Since "user difficulty," defined this way, is not directly observable from usability test data, it must be inferred from events that are directly observable in data, such as video and think-aloud audio recordings. Thus, an operational definition of "user difficulty" is needed. For the purposes of the present work, the criteria listed below were used by human usability experts to determine the onsets and offsets of periods of user difficulty. These criteria constitute an operational definition of "user difficulty." Note that even these relatively objective criteria have some room for subjectivity, e.g., deciding which statements constitute confusion. This subjectivity in determining user difficulty was measured by comparing the extent of agreement among multiple raters; section 4 contains details.

2.2.1 Criteria for onset of user difficulty

Five criteria were used to signal the onset of a period of user difficulty. These five criteria are directly observable events in video and think-aloud audio data. The five criteria are:

1. **User statements**: These occur when a user makes a statement or asks a question indicating confusion, frustration, uncertainty, lack of knowledge, or indecision. Such statements may start with phrases such as:
   - "I'm confused about ..."
   - "I'm not sure ..."
   - "I don't know ..."
   - "I can't figure out ..."
   - "I'm having a problem ..."
   - "I assume ..."
   - "How do I ..."
2. **Silence and inactivity:** This occurs when the user is silent *and* inactive with both the mouse and keyboard for at least 3 seconds. This 3-second threshold was arbitrarily chosen, but seemed to be the minimum amount of silence and inactivity that genuinely indicated confusion in the data used in this study. Silence or inactivity alone does not necessarily count as difficulty (unless accompanied by one or more of the other four criteria).
3. **Toggling:** This occurs when a user toggles an interface control, such as a checkbox, through one or more full cycles of its various states, without any intervening actions. An example of toggling is when a user checks, then unchecks, a checkbox. Toggling generally indicates that a user is confused about what the control does or about how to operate it.
4. **Help access:** This occurs when a user consults an online Help file. However, the difficulty is considered to have started in the period prior to consulting Help. The period begins on the click preceding the click that opens the Help window. The assumption is that users become confused in the period before consulting Help, so the click that brings up the Help window is too late to be considered the onset of the period of difficulty.
5. **Question to experimenter:** This criterion only applies to user tests in which a human experimenter is present, as happened in the present study. It occurs when the user asks a question of the experimenter about the experiment or the interface being tested.

2.2.2 Criteria for offset of user difficulty

Three criteria were used to signal the offset of a period of user difficulty. Like the criteria signaling the onset of user difficulty, these three criteria were observable events in video and think-aloud data. These three criteria were:

1. **Click:** This occurs when the user clicks on something with the mouse, which usually indicates that the user has moved on from the difficulty. There are two exceptions to using the "Click" criterion to end a period of user difficulty:

- If the "Toggling" criterion signals the onset of the period of difficulty, then the period does not end as long as the user is repeatedly clicking on the same control. The first click away from that control may signal the end of the difficulty.
- If the "Help" criterion signals the beginning of the period of difficulty, then the period does not end on clicks made while viewing the Help file; in this case, the "Help dismissed" criterion (see below) signals the end of the difficulty.

2. **User statement:** This occurs when the user makes a verbal assertion to do something, or when the user states that their confusion has been cleared up. User statements that signal the end of the difficulty may start with phrases such as:
   - "Ok, I'm going to hit the OK button ..."
   - "I'm going to try ..."
   - "All right, this looks right ..."
3. **Help dismissed:** If the "Help" criterion signals the start of the period of difficulty, then that period ends when the Help window is closed, minimized, or sent to background, i.e., when the user stops reading Help and starts doing something else.

2.2.3 User difficulty and interface defects

A hesitation detector does not detect interface defects directly; it detects periods of user difficulty that are the likely consequence of interface defects. Once hesitation-detector output has been obtained, a usability analyst must examine other sources of data, usually video and audio of user sessions, to determine which hesitations really indicate difficulty, as well as which particular defects caused each period of difficulty.

2.3 Accuracy

Accuracy is measured in terms of *hit rate* and *false-alarm rate*. Hit rate is the percentage of all periods of genuine user difficulty that the detector detects, while false-alarm rate is the percentage of events for which the detector incorrectly indicates user difficulty when none was present. Technical definitions of these terms are given below.

Because the input to hesitation detection is continuous in time, defining hit rate and false-alarm rate is not as straightforward as it is for detectors in which inputs are discrete. Thus, to make hit-rate and false-alarm-rate computation more straightforward, time is divided into discrete blocks. For each block, ground truth either designates the entire block as a period of user difficulty or a period of user non-difficulty. Also, for each block, the detector either classifies the block as a hesitation or a non-hesitation. A *hit* is any block that is designated as a period of user difficulty by ground truth and classified as a hesitation by the detector. A *false alarm* is any block that is designated as a period of user non-difficulty by ground truth but classified as a hesitation by the detector. *Hit rate* is defined as the number of hits divided by the number of blocks designated as periods of user difficulty by the ground truth. *False-alarm rate* is defined as the number of false alarms divided by the number of blocks designated as periods of user non-difficulty by the ground truth.

There are two reasonable ways to define a block:

1. A block is any contiguous portion of time during which the user experienced uninterrupted difficulty; under this definition, block length is variable, depending on the length of periods of user difficulty.
2. A block is a pre-defined, short amount of time, e.g., one second; under this definition, all blocks are the same length.

This paper uses both definitions of blocks. The former definition of blocks is used for hit-rate computation, while the latter is used for false-alarm-rate computation. The justification for using both definitions lies in the application of hesitation detection as a tool to help an analyst identify interface defects. In this application, it is not important that a hesitation lasts for the entire duration of a period of user difficulty; it is only important that a hesitation occurs *somewhere* within the period of difficulty, so that the difficulty will be brought to the analyst's attention. The analyst will then determine the full extent of the difficulty. Thus, so long as a hesitation occurs anywhere within a block of uninterrupted difficulty, that block is considered a hit. Alternatively, the penalty paid for a false alarm is that a usability analyst will have to examine only the portion of the data that constitutes the false alarm (plus, perhaps, a small amount of context); it is not necessary for the analyst to examine the entire period of contiguous non-difficulty. Thus, it makes sense to simply choose a small, uniform block size for computing false alarm rate. One second was chosen, because it is short enough that no major transitions from non-difficulty to difficulty occur within it.

2.4 Approach

The approach taken to answering the research question posed has the following four stages:

1. **Data collection:** Collect mouse, keyboard, video, and think-aloud audio data from users performing a task with either of two different user interfaces.
2. **Hesitation detection:** Use a hesitation detector to determine hesitations based on mouse and keyboard data.

3. **Ground-truth determination:** Have a usability expert determine ground-truth user difficulty from the video and audio data according to the criteria in section 2.2.

4. **Accuracy computation:** Compute the accuracy of hesitations as an indicator of user difficulty by comparing the hesitations with the ground truth.

Section 4 details the methodology.

3 Related work

Commonly used methods for user interface defect detection fall into four broad categories: inspection methods, model-based methods, surveys, and observation-based methods. Inspection methods, such as heuristic evaluation [20] and cognitive walkthrough [24], involve a usability expert reviewing an interface or interface specification for defects. They are particularly well-suited to finding defects in the early stages of the interface design cycle, because they do not require a working interface; they are notoriously inaccurate [3, 4, 12, 13]. Model-based methods, such as GOMS [9, 10], can give excellent predictions for skilled, expert performance with an interface, but they are not designed to detect aspects of an interface that may give novice or occasional users difficulty. Model-based methods also typically do not make predictions about where users, including experts, may make errors. Surveys are rarely applied to defect detection, but when they are, they tend to give subjective results that are difficult to quantify [5]. Observation-based methods have the advantages of providing direct data about what defects affect users, allowing for testing on both novices and experts, and giving quantitative results. The primary disadvantages of observation-based methods are that they generally require at least a prototype implementation of an interface, and as well as a great deal of analysis time.

Previous efforts have addressed the idea of using hesitations to determine periods of user difficulty. Maxion and deChambeau in 1995 were the first to propose using hesitation detection to spot user difficulty and identify user-interface defects [16]. They conducted a user study of an online library catalog application. The command-line-based application presented users with prompts for input, to which users responded with keystrokes. The time between prompt and response was measured, and times outside three standard deviations of the mean were reported as hesitations. The method yielded 66 hesitations from 12 participants. All hesitations were due to interface defects, so, remarkably, there were no false positives. However, no hit rate was reported because no ground truth was available. Furthermore, it is unclear how the results obtained for the command-line based library catalog application would generalize to today's common desktop applications, which are typically graphical user interfaces receiving continuous mouse and keyboard input.

In 1997, Maxion and Syme presented MetriStation, a software tool for user-study data collection and analysis, which included a component that analyzed the data for hesitations [18]. They made no attempt to measure the hesitation tool's accuracy, however.

Horvitz et al. in 1998 published their work on Lumiere, a system for inferring user goals and needs [7]. Lumiere employed a Bayesian statistical model to, in part, make inferences about when users needed assistance. The authors listed "introspection," which is defined similarly to "hesitation" in this paper, as one type of evidence used in their Bayesian network to infer that a user is having difficulty and needs assistance. The authors did not, however, measure the accuracy of either introspection or of the larger Bayes network as a means of detecting user difficulty.

A substantial amount of prior work has explored techniques other than hesitation detection to reduce the amount of analysis time required for observation-based interface evaluation. Ivory and Hearst [8] provide a thorough summary of this work. A few examples of the numerous tools for aiding usability data analysis include DRUM (Diagnostic Recorder for Usability Measurement) [15], AMME (Automatic Mental Model Evaluator) [22], and USINE (User Interface Evaluator) [14]. DRUM records significant user events, allows analysts to mark up significant points in usability-test video, and computes metrics like task time, time users spend having problems, and productive time. AMME constructs user mental models, interface state transitions, and quantitative metrics from log data. USINE takes as input a task model, and compares observed user actions to the task model to determine where user make errors. DRUM, AMME, and USINE are representative of numerous similar tools described by Ivory and Hearst.

4 Experimental method

As discussed in section 2.4, the approach encompassed four stages to address the question of hesitation detection accuracy: data collection, hesitation detection, ground-truth determination, and accuracy computation. The methods are explained below.

4.1 Data collection

Mouse and keyboard data streams, collected from a laboratory user study, were used as input to the hesitation detector evaluated in this work. Screen video and think-aloud audio from the same study were used by a human analyst to determine ground-truth periods of user difficulty. This section provides details on the user study from which these data were collected.

4.1.1 User interfaces tested

The data used for this work was collected as part of a laboratory user study comparing two interfaces designed for setting file permissions in Microsoft's Windows XP operating system [17]. The two interfaces tested were the native Windows XP file-permissions interface and an interface of the authors' design called Salmon. Both are graphical user interfaces, typical of the user interfaces of many of today's common desktop applications.

4.1.2 Participants

Twenty-three students and research staff members at Carnegie Mellon University participated in the study. Twelve participants were assigned to the XP interface, and 11 were assigned to Salmon. All participants were daily computer users, but they were only novice or occasional users of the file permissions interfaces. To establish that participants were novice or occasional users, they were asked to rate the frequency with which they set file permissions in their daily work, and to rate their familiarity with Windows file and folder security. Twenty of 23 participants reported setting file permissions a few times a month or less (the other three reported setting file permissions a few times a week or daily) and 22 out of 23 participants rated themselves "generally familiar" (10 participants), "vaguely familiar"(10 participants), or "unfamiliar" (2 participants) with Windows file and folder security. Only one participant rated himself "very familiar" with Windows file and folder security. No participants said their daily work involved Windows file and folder security. These answers suggest that the description of users as "novice or occasional" is accurate.

4.1.3 Task

Each participant completed seven file-permissions-setting tasks using the interface to which they were randomly assigned. Data from one of those tasks, called the Jack task, was used for this study. Of the seven tasks, the Jack task was chosen because it caused users the most difficulty.

Users were presented with task statements in a Web browser. They were able to consult the task statement at any time during their performance of a task. The task statement given to users for the Jack task was:

> The group ProjectE is working on projectEdata.txt, so everyone in ProjectE can read, write, or delete it. Jack (username: jack) has just been reassigned to another project and must not be allowed to change the file's contents, but should be allowed to read it. Make sure that effective now, Jack can read the file projectEdata.txt, but in no way change its contents.

Characteristics of the file-permissions task domain that may affect the results of the hesitation detector include:

- **Goal-oriented.** Users had a clear goal to accomplish, so this task was unlike, for example, browsing the Web or watching a video. Goal-oriented tasks require users to keep making progress; hesitations do not further task completion.
- **Very little typing.** At most, participants typed a few one-word usernames, so most hesitations detected in this study were hesitations in mouse usage.
- **Short time-to-completion.** Users took 169 seconds on average to complete the Jack task. The short task time allowed users to stay focused on the task, thus removing the potential for false alarms due to taking breaks, daydreaming, and the like.
- **Limited text to read.** Textual labels on the interfaces were limited to a few words (three or less), so the potential for false alarms due to users' reading long passages of text was minimal.

Many common tasks, such as system configuration, voting, and image manipulation, share these characteristics, so the results obtained by this study would be expected to generalize to a large class of interfaces and tasks.

4.2 Hesitation detection

A hesitation detector was implemented according to the algorithm sketched in section 2.1. The sensitivity parameter, which represents the minimum number of standard deviations a hesitation must be from the average pause length, was varied over a range of sensitivities from 0.5 to 24.0 in steps of 0.5. Mouse and keyboard logs for each user were provided as input to the detector.

4.3 Ground-truth determination

Ground truth was determined by a usability expert, who examined the video and audio logs collected from each of the 23 users during the user study. The criteria listed in section 2.2 (user statements, silence and inactivity, toggling, Help access, and questions to the experimenter) were used to identify periods of user difficulty. For each period of user difficulty, the onset and offset times were noted.

The expert's rating of ground-truth user difficulty was validated by two auxiliary raters' judgments of difficulty in 6 of the 23 data streams. This validation step was necessary because, although the criteria were designed to be as objective as possible, some subjectivity remains (most notably in the judgment of what user statements indicate confusion). The two auxiliary raters did not rate all 23 data streams because of the large time investment involved in doing so. However, the six data streams they did rate were balanced in their total length and in the interface used. Two were of short duration (less than 100 seconds), two were of medium

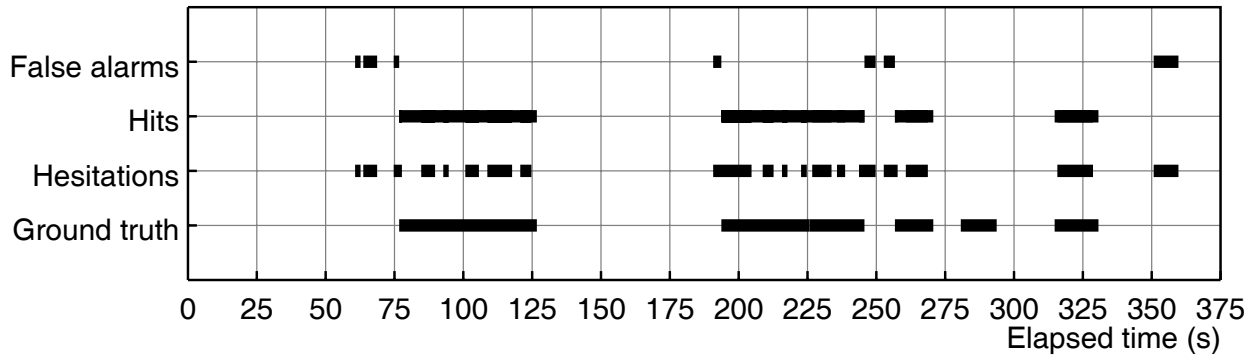

Figure 1: Timelines showing ground truth, hesitations, hits, and false alarms for one 364-second user session. The dark black regions indicate where these respective events occurred over the course of the 364 seconds.

duration (between 100 and 150 seconds), and two were of long duration (greater than 150 seconds). For each of these pairs of data streams, one was from an XP user and one was from a Salmon user. Within these constraints, the six data streams for validation were chosen at random. Three timelines, one from each rater, were laid out for each of the six validation data streams. The timelines were divided into discrete, one-second-long blocks, and a binary value was assigned to each block: 1 if the rater determined that the user was having difficulty during that block, and 0 if not. The auxiliary raters' judgments of difficulty were then compared, block-by-block, for agreement with the expert's ratings of difficulty. Within the six data streams, all three raters agreed for 68.9% of the data streams, while the expert had 80.0% agreement with the first auxiliary rater and 79.4% agreement with the second. For two raters, 70% agreement is generally considered acceptable, so the expert's judgment was deemed valid for determining ground truth.

4.4 Accuracy computation

Accuracy was measured in terms of hit rate and false-alarm rate. Recall the definitions of hit rate and false-alarm rate from section 2.3. The hit rate is defined as the percentage of all periods of user difficulty during which the hesitation detector finds any genuine hesitation. Note that, by this definition, the hesitation need not cover the entire duration of the difficulty period to be considered a hit; it is assumed that as long as some portion of the difficulty period is detected, a human analyst will find the full extent of the difficulty during the diagnosis stage. Note also that multiple hesitations may occur during the same period of difficulty; in this case, only one hit is counted. Hit rate, then, is simply the number of hits divided by the total number of distinct periods of difficulty identified by the expert. To measure false-alarm rate, two timelines were laid out – one representing the hesitation detector's output, and one representing ground truth. These timelines were divided into discrete, one-second-long blocks. Each block on the detector's timeline was assigned a binary value: 1, if the detector classified the block as part of a hesitation; 0, otherwise. Each block on the rater's timeline was also assigned a binary value: 1, if the rater designated the block as part of a period of user difficulty; 0, if not. A false alarm occurred whenever a block in the detector's timeline had a value of 1 but the corresponding block in the rater's timeline was 0. False-alarm rate was computed as the number of false alarms divided by the total number of blocks in the rater's timeline with a value of 0.

Figure 1 shows timelines for ground truth, hesitations (as output by the detector with sensitivity set to 2.0), hits, and false alarms for one 364-second user session. The figure shows five distinct periods of user difficulty consuming 145 seconds, indicated by dark black regions on the ground-truth timeline. Dark black regions on the hesitations timeline indicate regions the detector classified as hesitations. At least one hesitation coincided with each of four of the five periods of difficulty. These four periods of difficulty are marked as dark black regions on the hits timeline. For this user, the detector's hit rate is 4/5 = 80%. Twenty-nine one-second blocks were classified as hesitations by the detector, but designated as periods of non-difficulty in the ground truth, are marked as dark black regions on the false-alarms timeline. Since this user experienced 219 seconds of non-difficulty, the false alarm rate for this user is 29/219 = 13%.

5 Results

In the 23 data streams, there were 3389 total seconds of user data. According to the ground truth, users had difficulty during 999 of these 3389 seconds, or 29.5% of the time. There were 66 distinct periods of user difficulty, and

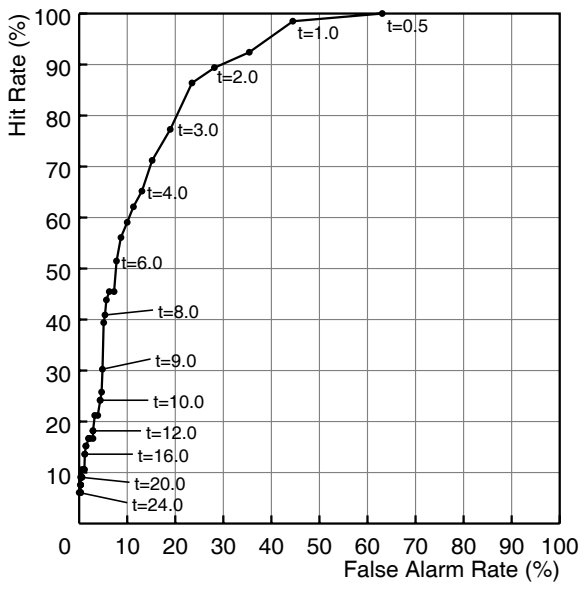

Figure 2: Receiver operating characteristic (ROC) curve for the hesitation detector applied to both Salmon and Windows interfaces' data. Points on the curve show detector accuracy at different sensitivities (t = threshold). The sensitivity represents the number of standard deviations beyond the mean a pause must be to be classified as a hesitation.

2390 seconds of non-difficulty. These latter two numbers are the denominators for the hit rate and false-alarm rate computations, respectively.

The receiver operating characteristic (ROC) curve in Figure 2 shows the main result of this paper. Each point on an ROC curve represents the accuracy of the hesitation detector for a particular value of the sensitivity parameter. The ROC curve shows how hit rate and false-alarm rate can be traded off, depending on the demands of a particular application. Representative points on the combined curve show that a 100% hit rate can be achieved if a 63% false alarm rate is tolerable, a 92% hit rate if a 35% false alarm rate is tolerable, an 86% hit rate if a 24% false alarm rate is tolerable, and all the way down to a 6.1% hit rate if no false alarms are tolerable. For some applications, a low false-alarm rate may be more important than detecting every period of user difficulty, while other applications may require detecting as many periods of difficulty as possible, regardless of false-alarm rate.

The ROC curve in Figure 2 shows results of the hesitation detector applied to both interfaces' data combined. ROC curves for the detector applied to the Salmon data and the Windows data separately are very similar to one another, as well as to the combined curve; they are not shown separately, because they would crowd the figure. The similarity of the curves for the two different interfaces suggests that the hesitation detection results are reasonably generalizable to different interface designs.

6 Discussion

Although the detector results may appear disappointing at first, they are in fact quite good when considered in the context of intended use. For example, analysts can spend less than 13% of typical analysis time, while still detecting more than 80% of the problem cases.

An example of hesitation detection applied to interface defect detection provides a concrete idea of how time can be saved by using hesitation detection. Suppose user data, namely mouse, keyboard, screen video, and think-aloud audio, have been collected from laboratory user-test sessions. If a usability analyst were to go through the video and audio media, searching for instances of user difficulty, it would take at least as long as the entire length of the media. (This is a very conservative estimate. In fact, Nielsen [19] estimates it takes 3 to 10 times as long as the entire length of the media; for this study, it took roughly 10 times as long as the media for the expert to generate ground truth.) Now suppose that instead of examining the entire media, a hesitation detector was applied, and the usability analyst only inspected those portions of the media flagged by the detector as potential instances of user difficulty. Suppose, also, that the analyst needed to watch 5 seconds preceding each period flagged by the detector to understand the context in which the hesitation occurred. (This 5-second figure is roughly what the authors have found is necessary in their own experience.) Although the hesitation detector may miss some instances of user difficulty, the same defect that caused a missed instance may be detected elsewhere. The measures of interest are the percentage of all defects detected and the amount of time saved by using the detector. Because some defects are manifested multiple times, and because of the 5 seconds needed by an analyst to gain context, the raw hit rate and false-alarm rate from the ROC curve do not quite give the full story.

Table 1 shows, for values of the sensitivity parameter from 2.0 to 6.0, the percentage of all defects detected (taking into account multiple manifestations of the same defect), the amount of time an analyst would save using the detector (taking into account the 5 seconds of prior context needed), and the hit and false-alarm rates from the combined ROC curve. It can be seen from the table, for example, that at a threshold value of 3.0, an analyst would spend 60.1% less time searching through areas of non-difficulty, while still finding 81.2% of the defects that would be found by watching the entire media. Again, this is making the very conservative estimate that an analyst would otherwise search through the entire media only once. If the analyst

Detector sensitivity	Percent defects detected	Percent time saved	HR	FAR
2.0	90.6	43.8	89.4	28.2
3.0	81.2	60.1	77.3	19.0
4.0	71.8	71.6	65.2	13.1
5.0	68.8	78.7	59.1	10.0
6.0	59.4	83.4	51.5	7.8

Table 1: Percentage of all distinct defects detected by the hesitation detector, percent of analyst time saved, Hit Rate (HR), and False Alarm Rate (FAR) for five values of the sensitivity (threshold) parameter (t).

would normally search through the media 3 to 10 times, time savings would be 87% to 96%, while still finding 81.2% of the defects.

These results show that using hesitation detection to sift through usability data can make user testing much less time-consuming for analysts. If desired, the analyst time saved could be spent by collecting and analyzing more data, which would likely make up for any defects lost due to misses by the hesitation detector. More data might also yield defects that would not have manifested in a smaller data set. If the percentage of defects detected still seems low, note that only a perfect analyst would detect 100% of all defects with only a single pass through the data; in reality, an analyst going through the data only once will miss numerous defects due to lack of attention and/or lack of vigilance. Moreover, the analyst's performance could be highly variable, while the detector's performance is repeatable.

Although a hesitation detector can be useful despite a high false-alarm rate, it is nevertheless worth looking at ways the false-alarm rate might be reduced. An informal look through the false alarms reported by the hesitation detector used in this study shows three top causes of false alarms: pauses while moving the hands from mouse to keyboard or keyboard to mouse, pauses while checking work just before committing changes, and pauses to read text such as unusually long interface labels, error messages, and Help files. A future version of the hesitation detector might apply heuristics to detect these causes of false alarms and thereby reduce the false-alarm rate.

A discussion of hesitation detection should also consider the method's limitations. First of all, hesitation detection will only catch interface defects that manifest as hesitations. Examples of defects that do not manifest as hesitations include defects that lead to errors of omission, defects that only cause brief delays (such as misspellings of labels), and defects that do not cause egregious hesitations (such as inconsistent fonts). Second, hesitation detection can only be applied after running user studies on fully implemented interfaces. Other usability methods, such as inspection-based methods, can isolate interface defects during the design stage, thus avoiding the expense of implementing a flawed design. Third, for some interfaces, such as Web browsers or multimedia players, hesitation detection might yield an unacceptably high false alarm rate because hesitations (to read text or watch video, for example) are inherently part of using the interfaces. Finally, hesitation detection can indicate defects, but does not suggest fixes for those defects.

7 Conclusion

Hesitation detection is a method for detecting instances of user difficulty, which are symptomatic of interface defects, in streams of data from user-interface test sessions. It can be applied to both field and lab-based user studies to save time that a usability analyst would otherwise have spent combing the data for trouble spots. This paper measured the accuracy of a hesitation detector for data from user tests of file-permissions-setting interfaces. The results show that hesitations are an effective means for detecting instances of user difficulty, and that hesitation detection promises to make usability studies less expensive and more comprehensive. For example, up to 96% of an analyst's wasted time can be saved by using hesitation detection, while still detecting 81% of all defects manifested in usability data.

8 Future work

In the present study, hesitation detection was evaluated in one task domain, namely setting file permissions. The file-permissions domain shares characteristics with many common task domains, such as system configuration, voting, and image manipulation, so the results obtained here are expected to generalize at least to those domains. Future work will test the method in these and other task domains, such as typing-intensive and long-duration tasks.

As discussed in section 6, the hesitation detection algorithm used in this study might be improved by employing heuristics to eliminate false alarms. A future version of the hesitation detection algorithm might exclude hesitations caused by transitions from mouse to keyboard or keyboard to mouse, pauses to read interface messages, and pauses to check work.

The study reported in this paper tested hesitation detection as a method for finding periods of user difficulty from logs of novice-user sessions. Future work might consider hesitation detection as a means to detect expert-user difficulty, which can indicate different interface defects from those indicated by novice-user difficulty.

This paper sought to establish hesitation detection as a potential method for saving usability analysts' time. The results reported here suggest that hesitation detection does

indeed have great potential to save analyst time. A planned follow-up study can compare the performance of analysts using hesitation detection against the performance of analysts using traditional techniques. This will determine the actual improvement that can be gained from hesitation detection, and will help to establish hesitation detection as a practical technique.

9 Acknowledgements

The authors would like to thank Rachel Roberts and Pat Loring for their contributions to this work. The authors also wish to thank the anonymous reviewers whose thoughtful remarks inspired improvements in the paper. This work was supported in part by National Science Foundation grant number CNS-0430474.

References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, January-March 2004.

[2] G. Cockton, D. Lavery, and A. Woolrych. Inspection-based evaluations. In J. A. Jacko and A. Sears, editors, *The Human-Computer Interaction Handbook*, chapter 57, pages 1118–1138. Lawrence Erlbaum Associates, Mahwah, NJ, 2003.

[3] G. Cockton and A. Woolrych. Sale must end: should discount methods be cleared off HCI's shelves? *Interactions*, 9(5):13–18, September 2002.

[4] H. W. Desurvire, J. M. Kondziela, and M. E. Atwood. What is gained and lost when using evaluation methods other than empirical testing. In *People and Computers VII: Proceedings of the HCI '92 Conference*, pages 89–102, Cambridge, September 1992. Cambridge University Press.

[5] J. S. Dumas. User-based evaluations. In J. A. Jacko and A. Sears, editors, *The Human-Computer Interaction Handbook*, chapter 56, pages 1093–1117. Lawrence Erlbaum Associates, Mahwah, NJ, 2003.

[6] K. A. Ericsson and H. A. Simon. *Protocol Analysis: Verbal Reports as Data*. MIT Press, Cambridge, MA, revised edition, 1993.

[7] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 256–265, Madison, WI, July 1988. Morgan Kaufmann.

[8] M. Y. Ivory and M. A. Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33(4):470–516, December 2001.

[9] B. E. John and D. E. Kieras. The GOMS family of user interface analysis techniques: comparison and contrast. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):320–351, December 1996.

[10] B. E. John and D. E. Kieras. Using GOMS for user interface design and evaluation: Which technique? *ACM Transactions on Computer-Human Interaction (TOCHI)*, 3(4):287–319, December 1996.

[11] D. Kieras. Model-based evaluations. In J. A. Jacko and A. Sears, editors, *The Human-Computer Interaction Handbook*, chapter 58, pages 1139–1151. Lawrence Erlbaum Associates, Mahwah, NJ, 2003.

[12] L.-C. Law and E. T. Hvannberg. Complementarity and convergence of heuristic evaluation and usability test: a case study of universal brokerage platform. In *Proceedings of the second Nordic conference on human-computer interaction*, pages 71–80, New York, NY, 2002. ACM Press.

[13] L.-C. Law and E. T. Hvannberg. Analysis of strategies for improving and estimating the effectiveness of heuristic evaluation. In *Proceedings of the third Nordic conference on human-computer interaction*, pages 241–250, New York, NY, 2004. ACM Press.

[14] A. Lecerof and F. Paterno. Automatic support for usability evaluation. *IEEE Transactions on Software Engineering*, 24(10):863–888, October 1998.

[15] M. Macleod and R. Rengger. The development of DRUM: A software tool for video-assisted usability evaluation. In J. Alty, D. Diaper, and S. Guest, editors, *People and Computers VIII: Proceedings of the HCI '93 Conference, September 1993*, chapter 20, pages 293–309. Cambridge University Press, Cambridge, 1993.

[16] R. A. Maxion and A. L. deChambeau. Dependability at the user interface. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing*, pages 528–535, Los Alamitos, CA, June 1995. IEEE Computer Society Press.

[17] R. A. Maxion and R. W. Reeder. Improving user-interface dependability through mitigation of human error. *International Journal of Human-Computer Studies*, 63(1-2):25–50, July 2005.

[18] R. A. Maxion and P. A. Syme. Metristation: A tool for user-interface fault detection. In *Twenty-Seventh International Symposium on Fault-Tolerant Computing*, pages 89–98, Los Alamitos, CA, June 1997. IEEE Computer Society Press.

[19] J. Nielsen. *Usability Engineering*. Academic Press, Inc., San Diego, CA, 1993.

[20] J. Nielsen. Heuristic evaluation. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, chapter 2, pages 25–62. John Wiley and Sons, New York, 1994.

[21] J. Nielsen and V. L. Phillips. Estimating the relative usability of two interfaces: Heuristic, formal, and empirical methods compared. In *INTERCHI '93 — Conference on Human Factors in Computing Systems*, pages 214–221, New York, NY, April 1993. ACM Press.

[22] M. Rauterberg. AMME: an automatic mental model evaluation to analyse user behaviour traced in a finite, discrete state space. *Ergonomics*, 36(11):1369–1380, November 1993.

[23] J. Rubin. *Handbook of Usability Testing*. John Wiley and Sons, Inc., New York, 1994.

[24] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practitioner's guide. In J. Nielsen and R. L. Mack, editors, *Usability Inspection Methods*, chapter 5, pages 105–140. John Wiley and Sons, New York, 1994.