MimEc: Intelligent User Notification of Faults in the Eclipse IDE

Lucas M. Layman, Laurie A. Williams, Robert St. Amant
Department of Computer Science
North Carolina State University
Campus Box 8206
Raleigh, NC 27695
lucas.layman@ncsu.edu, {williams, stamant}@csc.ncsu.edu

ABSTRACT
The earlier in the software process a fault is detected, the cheaper the cost of fixing the fault. Automated fault detection tools can provide developers with information throughout development, however, programming is a cognitively complex process and inundating the developer with information may do more harm than good. In this paper, we present MimEc, a part of the AWARE plug-in for the Eclipse integrated development environment. MimEc presents fault information to developers while they are writing code. The purpose of MimEc is to display only those faults in which a developer may be interested, thereby increasing the likelihood the developer will address the fault. MimEc infers interest in a fault based on fault criticality, relevance of the fault to the developer's current working context, and the developer's interactions with the programming environment. MimEc is currently under development and will be evaluated in both academic and professional settings.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]: Programming Environments – *integrated environments, interactive environments, programming workbench.*

General Terms
Human Factors

Keywords
Intelligent IDE; psychology of programming

1. INTRODUCTION
Research indicates that the time a software developer requires to fix a fault is positively correlated with *ignorance time,* the time between fault injection and the point at which the developer becomes consciously aware of a fault [3, 4]. *Automated fault detection* (AFD) tools can provide developers with prompt feedback on recently-introduced faults using static and/or dynamic analysis techniques, thereby reducing ignorance time. Some examples of AFD tools are FindBugs [8], Continuous Testing [13], and the continuous compilation feature of IDEs such as Eclipse.

Ideally, the software developer should act upon an *alert* – the notification of a potential fault – as soon as it is displayed. However, alerts that are provided but not acted upon may be an indication that the alerts are being produced too often, are not informative, and/or may be distracting to the developer. AFD tools also frequently provide alerts that are not actual faults in the system, thereby reducing the developer's confidence in the tool and increasing the likelihood that alert information will be ignored. Therefore, the designs goals of an AFD system should be to both provide fault information and ensure that the information is useful to the developer who is engaged in a development activity.

Programming tools specifically designed to tailor information to supplement developer's cognitive processes have enjoyed recent success [9, 12]. In this paper, we describe MimEc, a tool designed to align the benefits of automated fault detection with developers' cognitive needs and behaviors. MimEc, a plug-in for the Eclipse integrated development environment (IDE), presents fault information to developers while they are writing code. MimEc limits the display of fault information to those faults "of interest." MimEc determines user interest in a fault based on several factors, including fault criticality, relevance to the current working context, and whether the user is currently engaged in testing or debugging activity.

The remainder of this paper is organized as follows: Section 2 discusses the problem of task interruption; Section 3 describes the AWARE platform on which MimEc is based; Section 4 highlights the findings of our study on programmer fault-investigation behavior that serves as a basis for MimEc; Sections 5 provides an overview of the design of the MimEc system; and we conclude in Section 7.

2. TASK INTERRUPTION
From a psychological standpoint, our research is fundamentally a question of task interruption, memory, and attention. Limitations in human memory result in any interruption having the potential to interfere with and/or destroy a working task as elements of the interruption must take up some measure of working memory [5]. The degree to which an interrupting task interferes with a primary working task is dependent on the cognitive loads (the space requirements in working memory [6] and other factors. Debugging tasks (such as fixing a fault) have varying degrees of complexity that create different levels of cognitive load dependent on programmer expertise [1].

Interruption with regards to human-computer interaction has been studied, but typically not with regards to two related and dependent activities such as coding and debugging. Studies have

CHASE'08, May 13, 2008, Leipzig, Germany.

shown that interrupting task complexity and task similarity affect user performance [2]. McFarlane [11] asserts that a negotiated interruption style, where the system alerts the user but does not force attention away from the primary task (e.g. Microsoft Word's live spellchecker), is best in terms of user performance except in cases where the interrupting task is time-sensitive.

3. AWARE

Our platform for displaying alert information to the user is the AWARE plug-in for the Eclipse IDE. AWARE collects the output of third-party AFD tools (including FindBugs and the Eclipse compiler), estimates the severity of potential faults, ranks the faults according to the likelihood that it is *not* a false positive, and displays the alerts to the user [14]. A screenshot of AWARE can be seen in Figure 1. Each fault in the list contains the following information in order:

- a description of the problem, such as "possible null pointer" or "uninitialized variable"
- the folder, class file, and the line number at which the fault was detected in the code
- the probability at the fault is a true positive
- the severity of the fault from 1-3 with 3 being the most severe

In the AWARE display, users can double-click on an alert and they will be taken to the line of code where the alert indicates that a fault may exist. AWARE also provides facilities for users to filter alerts from the display if they determine the alert is a false positive. Filtering an alert will reduce the ranking of other alerts of the same time and in the same location at the method, class and package levels. For more information on the mechanics of the ranking algorithm, please see [14].

Our objective is to supplement AWARE with an intelligent interface component, MimEc. To make the alert information valuable to the developer, the information should align with developer's cognitive and behavioral needs. Consequently, the alerts presented to the user should be both "interesting" and informative for purposes of fault identification, and must also be presented in a non-disruptive manner. The first step in the development of MimEc was to understand what makes an alert "interesting" such that a developer will suspend his or her activities (immediately or in the future) to investigate the alert.

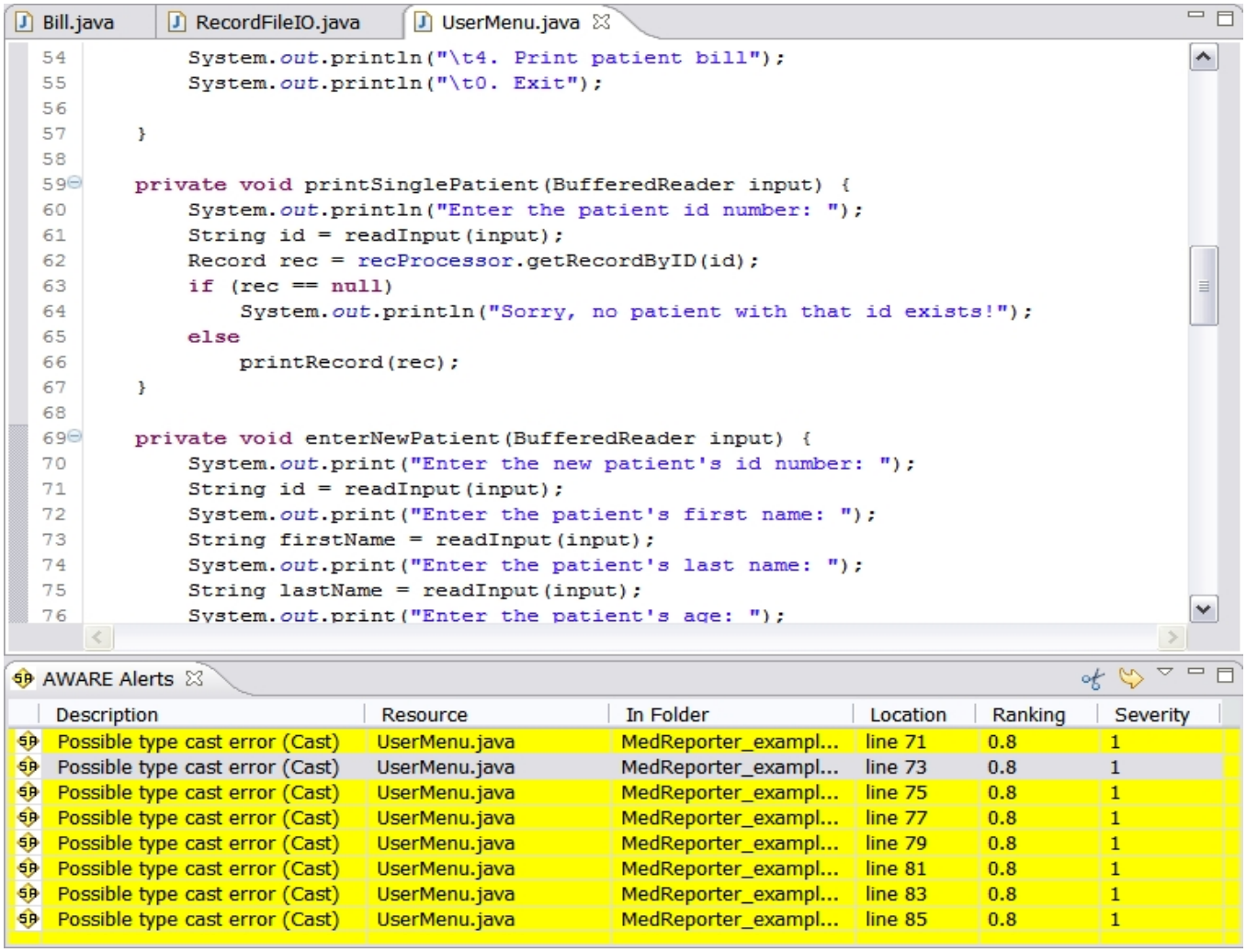

Figure 1. AWARE in the Eclipse IDE

4. DEVELOPER BEHAVIOR DURING FAULT INVESTIGATION

A controlled study was conducted with 18 developers of varying programming experience to discover why developers interrupt a programming task to debug a fault [10]. In understanding this behavior, our goal was to create an accurate model of user interest in alerts that would guide MimEc's intelligent interface. The study participants performed several programming tasks using the AWARE plug-in for Eclipse. During the programming tasks, AWARE notified the participants that potential faults were found in the code. The participants were asked to discuss the decision factors that influenced whether or not they address the alerts. The study sessions were audio recorded, transcribed and coded for analysis.

The coding process yielded 37 distinct themes organized into seven categories dealing with task interruption and fault assessment:

1. Strategies – describe developer behavior as relates to addressing alerts
2. Fault assessment criteria – the factors used by developers to determine whether or not to interrupt the primary task to address an alert
3. Interruption points – specifically when in time the primary task will be interrupted
4. Environment – influences created by the programming environment itself
5. Individual differences – attributes of the developers
6. Perspectives – the impacts of developer understanding of the example program or AWARE tool that influenced interruptions
7. External influences – factors related to the experimental setup that influenced developer behavior

The findings of our study lead to several conjectures about the design of the MimEc intelligent interface [10]. MimEc should present alert information that is relevant to the primary programming task with accurate and precise descriptions. The alert severity and the timing of alert notification should be customizable. Also, the AFD tool must be accurate and reliable to build trust with the developer.

The findings of the study also identified two factors that MimEc uses to determine whether or not to display alert information to the user: 1) the *criticality* of the alert; and 2) the *relevance* of the alert to the user's primary task. Criticality, the potential impact of a fault, was assessed through a combination of alert description, ranking and severity. This alert information is provided by the AWARE system. Relevance is a less well-defined concept, but can be inferred through observing the user's interactions with the IDE and through programmatic means such as impact analysis.

5. DESIGN OF THE MimEc SYSTEM

This section describes the design considerations and design plans of the MimEc system. The design is based on the results of our formative study and published recommendations for the design of mixed-initiative systems [7]. The system is still under development and so only limited technical information is available in this paper.

5.1 Recording user interactions

MimEc records the user's interactions with the Eclipse IDE through the monitoring facilities of another plug-in, Mylyn (formerly Mylar) [9]. Interaction events are recorded when the user selects certain elements in the IDE, including lines of source code, icons representing classes in the navigation browser, alerts in the AWARE windows, preference pages, and many more. The interaction events contain information regarding which object was selected by the user and a timestamp of the selection. For example, if the user selects a line of code, MimEc will record that the selection took place in the Java editing window, the name of the method, class, and package name of the selected line of code, and the time of the selection. The interaction events form a history of the user's interactions with the IDE over time.

The history of user interactions is useful for several reasons. Event sequences can identify various programming activities, such as "searching for a method name," "running a test case," or "changing the runtime environment." Collections of these activities can reveal patterns that are indicative of usage modes, such as "development mode" or "testing/debugging mode." Developer goals with regards to fault notification may change depending on his or her current activities, and the system should account for this in some way. For example, a developer writing new code may be less inclined to investigate alerts, whereas a developer running tests or removing compiler warnings may be more inclined to fix potential faults in the system. The interaction history also can be used to help infer the relevance of an alert to the user's current working context. If an alert appears in a class, method, or other artifact with which the user has recently interacted, this may be an indication that the alert is relevant to the user's current working context.

We will distribute an initial version of MimEc that will collect user interaction data from students in a software project course. This initial version will not interact with the users, but will collect users' interactions with the IDE and with the faults provided by AWARE. We will also use screen capture software to video record students as they are developing in Eclipse while MimEc is collecting interaction data. We will couple these screen recordings with researcher observations of the students' programming activities and use this combined information to help connect MimEc's interaction event data with actual development activities.

As development on the MimEc system continues, we will employ machine learning algorithms to identify interaction patterns that can be associated with usage modes and relevant context. This initial data set will provide the foundation for creating a decision model that reflects a developer's interest in an alert. Machine learning facilities will also be distributed with the MimEc system to adjust the baseline model to the usage patterns of individual users and to differing system designs.

5.2 Inferring user interest

Our initial usage data set will be used to help build a degree of interest model (DOI) that infers the user's interest in an alert using three basic components: 1) usage mode, 2) alert criticality (ranking, severity, and type), and 3) alert relevance. The usage mode is inferred from the interaction history, or can be explicitly set by the user. Alert criticality is computed from information furnished by the AWARE system. Alert relevance is inferred from the user interaction history and some system analysis.

Since interest may be calculated on thousands of alerts, all analysis and computations must be lightweight.

Once an estimate of user interest in an alert is made, MimEc must decide if the interest level is high enough to interrupt the user by drawing attention to that alert. Horvitz's discussion of decision theory as related to intelligent agents [7] serves as the basis for determining this threshold. If the utility threshold is met, MimEc displays the alert in the AWARE view (or in another to-be-determined, user-specified manner).

5.3 User feedback and control

The behavior of MimEc is customizable by the user. Users can control the style of alert notifications (background color, source code highlighting, pop-up notifications). Users can change the utility function thresholds for the different usage modes. Also, users are able to set explicit priorities for different alert types (such as null pointer warnings) to influence the DOI model directly.

MimEc's DOI model is guided also by user feedback. A user selecting an alert displayed by MimEc suggests that the DOI model was correct in actively displaying the alert, whereas a user filtering the alert is a clear indication that the alert is not of interest in the current usage mode. This feedback will be used to tune the parameters of the DOI model using an adaptive learning mechanism similar to that used to create the baseline DOI model. Feedback-based adaptation will enable the DOI model to more closely reflect the behaviors of individual developers and adjust to the fault distributions of a variety of project domains and environments.

6. FUTURE WORK AND CONCLUSION

The MimEc system is still under development, though we will have a fully functional version integrated with the Eclipse IDE at the time of the workshop. The system will be deployed to students for data collection, and to both students and professional for evaluation in the upcoming months. MimEc will be evaluated primarily on two dimensions: 1) accuracy of the prediction of user interest in an alert, and 2) whether inference of user interest lessens the time faults are latent in the system. We also expect to gain insight into the usage patterns of IDEs in our user population through our logging of interaction events. The usage information alone may be of use to the IDE designers and to parties interested in developer behavior.

The overall goal of the MimEc system is to present fault information that developers are interested in while they are coding, increasing the likelihood the developers will act on the fault. Research suggests [3, 4] that the overall cost of fixing the fault will be reduced by shortening the time the fault is latent in the system. We have performed a controlled study to understand developer decision-making behavior with regards to alert investigation. In creating a system that accounts for developer behavior and preference, our aim is to make a tool that provides developers with useful fault information in a way that aligns with their own behaviors.

ACKNOWLEDGEMENTS

This material is based upon the work supported by the National Science Foundation under Grant ITWF 00305917. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. REFERENCES

[1] B. Adelson, D. Littman, K. Ehrleich, J. Black, and E. Soloway, "Novice-Expert Differences in Software Design," in *Human-Computer Interaction (INTERACT '84)*, 1984, pp. 187-192.

[2] B. P. Bailey, J. A. Konstan, and J. V. Carlis, "Measuring the Effects of Interruptions on Task Performance in the User Interface," in *IEEE International Conference on Systems, Man, and Cybernetics 2000 (SMC '00)*, Nashville, TN, 2000, pp. 757-762.

[3] W. Baziuk, "BNR/NORTEL: Path to improve product quality, reliability, and customer satisfaction," in *International Symposium on Software Reliability Engineering*, Toulouse, France, 1995, pp. 256-262.

[4] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[5] M. B. Edwards and S. D. Gronlund, "Task Interruption and its Effects on Memory," *Memory*, vol. 6, 1998 pp. 665-687.

[6] T. Gillie and D. Broadbent, "What Makes Interruptions Disruptive? A Study of Length, Similarity, and Complexity?," *Psychological Research*, vol. 50, 1989 1989 pp. 243-250.

[7] E. Horvitz, "Principles of Mixed-Initiative User Interfaces," in *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*, Pittsburgh, PA, 1999, pp. 159-166.

[8] D. Hovemeyer and B. Pugh, "Finding Bugs is Easy," *ACM SIGPLAN Notices*, vol. 39, December 2004 pp. 92-106.

[9] M. Kersten and G. C. Murphy, "Using Task Context to Improve Programmer Productivity," in *Foundations of Software Engineering (SIGSOFT/FSE '06)*, Portland, OR, 2006, pp. 1-11.

[10] L. Layman, L. Williams, and R. St. Amant, "Toward Reducing Fault Fix Time: Understanding Developer Behavior for the Design of Automated Fault Detection Tools," in *Empirical Software Engineering and Measurement (ESEM '07)*, Madrid, Spain, 2007, pp. 176-185.

[11] D. C. McFarlane, "Comparison of Four Primary Methods for Coordinating the Interruption of People in Human-Computer Interaction," *Human-Computer Interaction*, vol. 17, 2002 2002 pp. 63-139.

[12] B. A. Myers, D. Weitzman, A. J. Ko, and D. H. Chau, "Answering Why and Why Not Questions in User Interfaces," in *ACM Conference on human Factors in Computing Systems*, Montreal, Canada, 2006, pp. 397-406.

[13] D. Saff and M. D. Ernst, "Continuous Testing in Eclipse," in *27th International Conference on Software Engineering (ICSE '05)*, St. Louis, MO, 2005, pp. 668-669.

[14] S. E. Smith, L. Williams, and J. Xu, "Expediting Programmer AWAREness of Anomalous Code," in *International Symposium on Software Reliability Engineering (ISSRE '05)*, Chicago, IL, 2005.