Towards Identification of Software Improvements and Specification Updates By Comparing Monitored and Specified End-User Behavior

Tobias Roehm, Bernd Bruegge
Technische Universität München
Munich, Germany
{roehm, bruegge}@in.tum.de

Tom-Michael Hesse, Barbara Paech
University of Heidelberg
Heidelberg, Germany
{hesse, paech}@informatik.uni-heidelberg.de

***Abstract*—Support of end-user needs is an important success factor for a software application. In order to optimize the support of end-user needs, developers have to be aware of them and their evolution over time. But a communication gap between developers and users leads to ignorance of developers about how users use their application. Also, developer assumptions about user behavior are rarely tested and corrected if they are wrong. Consequently, many software applications have a mediocre support of user needs and user problems as well as changes in user needs are detected rather late.**

In this paper, we present a research agenda addressing this problem by comparing use case descriptions to monitored user actions. More specifically, we propose to monitor user actions using instrumentation, detect the current use case of a user using machine learning, and compare use case steps to monitored user actions. By detecting differences between both, we identify mismatches between user behavior and developer assumptions reflected in use case descriptions. Those mismatches can serve as starting points to identify software improvements, to test the use case specification and identify updates, and to revise training programs. Finally, we sketch a plan to evaluate our approach.

***Keywords*—User needs, User monitoring, Use case detection, Comparison of observed and specified behavior, Specification testing, Reverse modeling, Machine learning, Software evolution**

I. Introduction

In a competitive software market end-users can choose among several rival applications and pick the application with the best support of their needs. The success of a software system in such a situation is determined by the acceptance of its end-users. Consequently, developers should cooperate with end-users to understand user needs, develop software that supports those needs, and ensure the continuous support during software evolution.

Unfortunately, a communication gap between end-users and developers exists [12] that hinders cooperation between developers and users. Developers might not have access to end-users, end-users might not be willing or able to express their needs and problems, or developers and end-users might misunderstand each other. The result of this gap is ignorance at the developers' side about how end-users use the application [19]. Also, developer assumptions about end-user behavior are seldom tested and corrected if they are wrong. Consequently, many software applications have a mediocre support of user needs and user problems as well as changes in user needs are detected rather late.

In this paper, we propose an approach to address this problem. More specifically, we propose to instrument software applications to monitor user actions. Further, we propose to detect the use case a user is currently performing and compare its flow of events to monitored user actions. We propose to consider use cases as they document the assumptions of developers about how a user will use an application in a fine-grained, detailed way. We argue that detected differences between user actions and use case steps can be exploited in several ways. First, a difference can serve as starting point for further exploration and finally identify software improvements in terms of functionality or usability. Second, the use case specification can be tested based on the observation (specification testing) and updates can be identified if the use case specification is incorrect or incomplete (reverse modeling). Third, training programs for users can be designed or refined based on knowledge about application usage and difficulties users are facing. As developer assumptions about application usage are reflected and documented in the use case description, our approach allows comparing the assumptions of developers about application usage to actual usage obtained from monitored user actions. Our approach is complementary to existing approaches such as feedback mechanisms, on-site user observations, usability labs or user workshops.

The contributions of this paper are the following. First, we propose to compare monitored user actions to the use case specification of an application and sketch a corresponding approach. Second, we discuss how these differences can be exploited to identify software improvements, to test and update the use case specification, and to revise training programs for users.

This paper is organized as follows: In Section II we review related work. In Section III we describe our approach in detail. In Section IV we sketch an evaluation strategy for our approach and finally sum up in Section V.

II. Related Work

Maalej et al. [12], [13] described the problem of communication gaps between developers and end-users and proposed to consider user input as important information. We instantiate their generic framework and additionally propose to compare specified and monitored user behavior. Kim et al. [10] describe TRUE, an approach to collect user actions by instrumentation and exploit this information to improve video games. While their approach and goal is similar to ours, they do not compare monitored and specified behavior automatically.

a) Use Case Mining: El-Ramly et al. [3], [5], [6] developed an approach to recover use cases from monitored user actions and exploit that knowledge in user interface reengineering. Similarly, Antonio et al. [1] and Li et al. [11] developed approaches to recover a use case model from runtime information. While these approaches overlap with ours by the use case detection, they do not compare monitored user actions with specified use cases. We plan to reuse their work for use case detection.

b) Comparison of Monitored and Specified Behavior: Comparing monitored user behavior to a specification of expected user behavior has been studied by other researchers. Paternò et al. [15] propose an approach using task models, i.e. a hierarchical decomposition of a task, to capture expected behavior of users and compare them to monitored behavior. Feuerstack et al. [7] use UI models to generate user interfaces automatically and evaluate their usability by comparing monitored user behavior to the original models. We are currently evaluating whether and how we can reuse their work.
Robinson [17], [18] proposes an approach similar to ours. He monitors user and system behavior and compares it to user goals specified by OCL-like statements. While Robinson focuses on abstract goals, we monitor and compare lists of user actions.

c) Automated Usability Testing: Several approaches have been proposed to monitor user actions and exploit them to (semi-) automate usability testing. Ivory and Hearst [8] review the state of the art of automated usability evaluation. Tao [21] proposes an approach to capture user interactions and analyze usability in early development phases. Several approaches have been proposed to evaluate the usability of web applications [2], [15] as well as mobile applications [9], [16]. Those approaches focus on usability evaluation while we do not only target usability problems but also discuss specification testing and updating.

III. Our Approach

In this section we describe our approach in detail.

A. Motivating Example

In order to motivate our approach, we start with an example that is depicted in Figure 1. We consider the development of online banking software. Before the implementation of the software, developers talked to banking customers and identified the use case steps (see the left hand side of Figure 1). Now the software is implemented as a web application, deployed

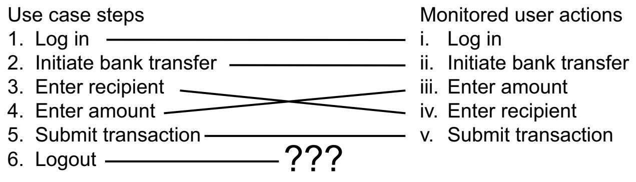

Figure 1. Bank Transfer Example

and end-users use it to do their online banking activities. A sensor within the web server monitors HTTP requests and identifies user actions from the URLs requested. For example, the initiation of a new bank transfer can be detected when a user initiates a HTTP request to the url /BankTransfer.do. All actions of a user are monitored (see the right hand side of Figure 1). When comparing use case steps and user actions, we detect that entering recipient data and amount are switched in order and the user forgot to logout, e.g. by closing the browser or surfing to another web page. These differences are presented to the developers of the banking application and they have to decide how to handle them. They decide to ignore the switched order of data entry and to add an automatic logout feature after a certain inactivity time of a user - a standard feature of online banking applications today.

B. Research Questions

We address the following research questions.

- How can semantically meaningful user actions be obtained (RQ 1)?
- How can the current use case a user is performing be identified based on traces of monitored user actions (RQ 2)?
- How can use case steps and monitored user actions be compared (RQ 3)?
- How can differences between use case steps and monitored user actions be exploited (RQ 4)?

Regarding RQ1, monitoring of low-level user actions such as mouse clicks and text entered is easy, but we need semantically meaningful user actions on a similar level of abstraction as use case steps to be able to compare both. Further, we argue that detecting differences as such is not a worthwhile goal in itself but the exploitation of the knowledge gained to support evolution decisions and test developer assumptions about application usage (RQ 4).

C. General Framework

In this section we sketch a framework that is necessary to monitor user actions and compare use case steps to monitored user actions. Figure 2 gives an overview of our framework. A user interacts with an application that is instrumented with one or more sensors. Sensors can be implemented using different implementation approaches - framework hooks, log file monitors, special monitoring code, or byte code instrumentation. Also, they can target different frameworks and hence exhibit a varying degree of application independence and reusability -

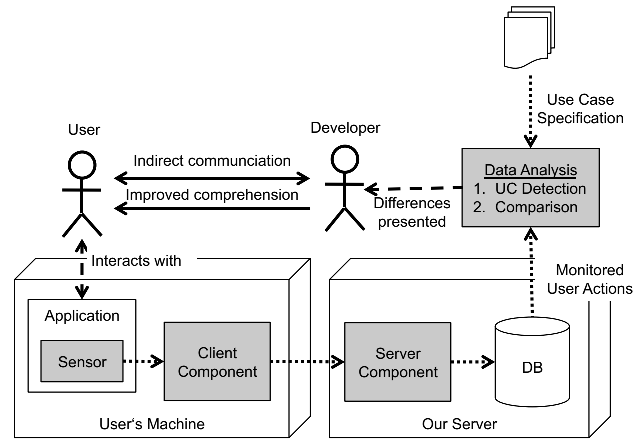

Figure 2. Framework Overview
Grey boxes denote components of our framework.

from being completely independent of the application (e. g. a sensor monitoring log files), via enabling reuse in the same framework (e. g. a sensor for RCP framework) to being completely application dependent (e. g. a sensor integrated in the application source code). A sensor monitors user actions and sends them to a client component running on the user's machine, too. The client component processes and aggregates the user actions and sends them to a server component, which stores them in a database. This architecture provides the possibility to process user actions on the client side and collect additional context information while performing the main processing on the server side to minimize the overhead on a user's machine. The monitored user actions from the database are analyzed in two ways. First, the use case performed by the user among the set of specified use cases is detected. Second, the flow of events of the current use case is compared to monitored user actions and detected differences are presented to developers. Overall, we achieve an indirect, automated communication which is effortless for the user.

D. Detection of the Current Use Case

We are exploring machine learning algorithms to detect the current use case a user is performing based on his or her actions. As we can observe user actions directly but use cases not, Hidden Markov Models seem to be appropriate for our case. Saito et al. [20] developed an interesting, related approach targeting user tasks. Further, El-Ramly et al. [4]–[6], Antonio et al. [1], and Li et al. [11] did work on analyzing user actions and mine use cases models that we plan to reuse.

E. Comparison of Use Cases and Monitored User Actions

In this section we describe the steps necessary to compare the flow of events of a use case to monitored user actions.

1) Abstracting Monitored User Actions: In order to be able to compare use case steps and monitored user actions directly, they have to have a similar level of abstraction. We are experimenting with three strategies to abstract monitored user actions. First, our sensors monitor user actions already at a high level of abstraction, i.e. not every mouse or key action but manipulation actions that consist of several mouse and key actions. Second, we use sequential pattern mining to detect frequent patterns in traces of monitored user action. When a frequent user action pattern is detected, the sequence of user actions forming this pattern can be replaced in the traces by a new, single, and more abstract user action. Third, we use a taxonomy of user actions in order to reason about user actions and abstract user action types.

2) Mapping between Use Case Steps and User Actions: In order to be able to compare use case steps and monitored user actions a mapping between both has to be established. This mapping establishes a relationship from use case steps to types of user actions. It denotes the relationship "This use case step can be represented by this type(s) of user action". We are experimenting with two strategies to accomplish this mapping. In the first strategy, a developer assigns each use case step to one or more user action types that correspond to it. In the second strategy, we compare the similarity of the textual description of a use case step to the textual description of a user action in our user action taxonomy. If the textual similarity exceeds a threshold parameter, we map the corresponding user action to a use case step.

3) Comparing Use Case Steps and User Actions: Finally we compare a trace of monitored user actions to the flow of events of the current use case based on the mapping described above. We expect the differences to be additional/ missing steps and a different order of steps. Hence, we iterate over the steps of the current use case and check for each step whether a corresponding user action or user action pattern was monitored. Also, we check if there is a difference in the order.

F. Exploitation of Detected Differences

We argue that differences between use case steps and user actions are not good or bad in itself. Developers have to analyze them, determine how to handle them, and decide if any of the following situations applies. First, a detected difference can be the starting point for further analysis and finally trigger an improvement of the application. Examples of improvements are additional functionality, changed functionality, or removal of a usability problem. Second, if the monitored user behavior reflects a valid use of the application that is not documented in the current specification, the specification should be updated based on the observation. Third, a detected difference can trigger changes and tailoring of user training by taking information about user behavior and problems users are facing into account. Overall, a detected difference reveals a wrong assumption of developers (who wrote the use cases) about application usage and might trigger further investigation. Thereby, it helps developers to comprehend the behavior of users and improve the support of user needs within their application.

IV. Evaluation Strategy

In this section we outline our plan to evaluate our proposed approach. We already implemented the monitoring part of the framework described in Section III. We implemented sensors that track user actions in Eclipse RCP applications and J2EE-based web applications [14]. Using these sensors we collected user action traces of five users that worked with an instrumented application for several weeks.

We plan to evaluate our approach with two case studies. In the first case study, we will monitor user actions using our sensors and ask users to create a protocol of the tasks/ use cases they are performing. The obtained dataset allows us to learn predictors of the current use case and evaluate their prediction accuracy. In the second case study, we will monitor user actions using our sensors, compare them to the flow of events of the current use case, and show detected differences to developers. We will evaluate our approach in three directions. First, we evaluate its correctness, i.e. if the detected differences are real differences. Second, we evaluate its helpfulness for developers, i.e. if the detected differences help developers to improve their application or update the use case specification. Third, we evaluate its performance overhead, i.e. how much overhead our sensors introduce and if the overhead disturbs users in their daily work.

In the case studies we will use two real-world applications and their users and developers. The first application will be UNICASE[1], an Eclipse-based CASE tool that supports UML modeling, rationale-based software engineering, and capture and use of project knowledge. The second application will be an application developed by a partner software company.

V. Summary

In this paper we proposed an approach to detect the current use case a user is performing and to compare its flow of events to monitored user actions. Detected differences can be exploited to identify software improvements, to test the use case specification and identify updates, and to revise training programs for users. Overall, we aim to improve user comprehension on the developer's side and thereby help developers to maximize the support of user needs in their applications.

Our next steps will be to continue implementation of our approach and evaluate it by case studies with two real-world applications and their users and developers. Further, we will investigate the impact of privacy issues on our approach.

Acknowledgements

This work was partially supported by the DFG (German Research Foundation) under the Priority Programme SPP1593: Design For Future – Managed Software Evolution and the European Commission (FastFix project, grant FP7-258109).

[1]http://code.google.com/p/unicase

References

[1] G. Antonio, D. Lucca, A. R. Fasolino, U. D. Carlini, N. Federico, and V. Claudio. Recovering use case models from object-oriented code: a thread-based approach. In *Proc. of the Seventh Working Conf. on Reverse Engineering*, pages 108–117. IEEE, 2000.

[2] R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move - User activity tracking for website usability evaluation and implicit interaction. In *Proc. of the 15th Int. Conf. on World Wide Web*, pages 203–212. ACM, 2006.

[3] M. El-Ramly and E. Stroulia. Mining software usage data. In *Proc. of the 1st Int. Workshop on Mining Software Repositories*, MSR 2004, 2004.

[4] M. El-Ramly, E. Stroulia, and P. Sorenson. From run-time behavior to usage scenarios: An interaction-pattern mining approach. In *Proc. of the Eighth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, KDD'02, pages 315–324. ACM, 2002.

[5] M. El-Ramly, E. Stroulia, and P. Sorenson. Mining system-user interaction traces for use case models. In *Proc. of the 10th Int. Workshop on Program Comprehension*, pages 21 – 29. IEEE, 2002.

[6] M. El-Ramly, E. Stroulia, and P. Sorenson. Recovering software requirements from system-user interaction traces. In *Proc. of the 14th Int Conf. on Software Engineering and Knowledge Engineering*, SEKE '02, pages 447—-454. ACM, 2002.

[7] S. Feuerstack, M. Blumendorf, M. Kern, M. Kruppa, M. Quade, M. Runge, and S. Albayrak. Automated usability evaluation during model-based interactive system development. In *Engineering Interactive Systems*, volume 5247 of *LNCS*, pages 134–141. Springer, 2008.

[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, 2001.

[9] D. Kim and K.-p. Lee. Development of interactive logger for understanding user's interaction with mobile phone. In *Human-Computer Interaction. Interaction Platforms and Techniques*, volume 4551 of *LNCS*, pages 394–400. Springer, 2007.

[10] J. H. Kim, D. V. Gunn, E. Schuh, B. C. Phillips, R. J. Pagulayan, and D. Wixon. Tracking real-time user experience (TRUE): A comprehensive instrumentation solution for complex systems. In *CHI 2008 Proceedings*, pages 443–451. ACM, 2008.

[11] Q. Li, S. Hu, P. Chen, L. Wu, and W. Chen. Discovering and mining use case model in reverse engineering. In *Proc. of the 4th Int. Conf. on Fuzzy Systems and Knowledge Discovery*, FSKD'07. IEEE, 2007.

[12] W. Maalej, H. Happel, and A. Rashid. When users become collaborators: Towards continuous and context-aware user input. *Proc. of the 24th ACM SIGPLAN Conf. Comp. on Object Oriented Programming Systems Languages and Applications*, pages 981–990, 2009.

[13] W. Maalej and D. Pagano. On the socialness of software. In *Ninth IEEE Int. Conf. on Dependable, Autonomic and Secure Computing*, DASC, pages 864–871. IEEE, 2011.

[14] D. Pagano, M. Juan, A. Bagnato, T. Roehm, B. Bruegge, and W. Maalej. FastFix: Monitoring control for remote software maintenance. In *ICSE 2012 Proceedings*, pages 1437–1438. IEEE, 2012.

[15] F. Paternò, A. Piruzza, and C. Santoro. Remote web usability evaluation exploiting multimodal information on user behavior. In *Computer-Aided Design of User Interfaces V*, pages 287–298. Springer, 2007.

[16] F. Paternò, A. Russino, C. Santoro, and V. G. Moruzzi. Remote evaluation of mobile applications. In *Task Models and Diagrams for User Interface Design*, volume 4849 of *LNCS*, pages 155–169. Springer, 2007.

[17] W. N. Robinson. Seeking quality through user-goal monitoring. *IEEE Software*, 26(5), 2009.

[18] W. N. Robinson. A roadmap for comprehensive requirements monitoring. *IEEE Software*, 43(5):64–72, 2010.

[19] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej. How do professional developers comprehend software? In *ICSE 2012 Proceedings*, pages 255–265. IEEE, 2012.

[20] R. Saito, T. Kuboyama, Y. Yamakawa, and H. Yasuda. Understanding user behavior through summarization of window transition logs. In *Databases in Networked Information Systems*, volume 7108 of *LNCS*, pages 162–178. Springer, 2011.

[21] Y. Tao. Capturing user interface events with aspects. In *Human-Computer Interaction. HCI Applications and Services*, volume 4553 of *LNCS*, pages 1170–1179. Springer, 2007.