Contents lists available at ScienceDirect

Int. J. Human–Computer Studies

journal homepage: www.elsevier.com/locate/ijhcs

Automatic detection of usability smells in web applications

Julián Grigera[a,*], Alejandra Garrido[a,b], José Matías Rivero[a,b], Gustavo Rossi[a,b]

[a] LIFIA, Facultad de Informática, UNLP, La Plata, Argentina
[b] CONICET, Argentina

ARTICLE INFO

Keywords:
Usability testing
Web-based interaction
Refactoring
Log analysis

ABSTRACT

Usability assessment of web applications continues to be an expensive and often neglected practice. While large companies are able to spare resources for studying and improving usability in their products, smaller businesses often divert theirs in other aspects. To help these cases, researches have devised automatic approaches for user interaction analysis, and there are commercial services that offer automated usability statistics at relatively low fees. However, most existing approaches still fall short in specifying the usability problems concretely enough to identify and suggest solutions. In this work we describe usability smells of user interaction, i.e., hints of usability problems on running web applications, and the process in which they can be identified by analyzing user interaction events. We also describe USF, the tool that implements the process in a fully automated way with minimum setup effort. USF analyses user interaction events on-the-fly, discovers usability smells and reports them together with a concrete solution in terms of a usability refactoring, providing usability advice for deployed web applications.

1. Introduction

Web applications help us in many of our daily life activities, like shopping, news reading, social interaction, home banking, trip planning or requesting a doctor's appointment. Every day new websites appear broadening our possibilities to accomplish tasks comfortably from home, and yet many times they suffer from usability problems that make them awkward and hard to use. As Nielsen said, *usability rules the web*, and it is crucial for a website's success (Nielsen and Loranger, 2006; Gregg and Walczak, 2010). Companies acknowledge that competition is so high that they will hardly survive if they do not invest in usability, although it still remains expensive and therefore neglected despite the progress in research and tools (Nielsen and Loranger, 2006).

One of the most popular ways of evaluating usability is by conducting usability tests, particularly, user tests (Rubin and Chisnell, 2008). The benefit of user testing over inspection methods like heuristic evaluations is that it captures real usage data and users' experiences. The down-side, however, is that it requires recruiting users and spending time and resources for experts first to design the tests and afterwards to analyze the results, discover the problems and find solutions for those problems.

To overcome the need of having the expert manually collecting and comparing test results, different automated approaches exist for remote user testing. Several approaches log user interaction (UI) events and perform some log analysis to help the expert discover usage patterns (Santana and Baranauskas, 2015). The results are usually presented with sophisticated visualization tools that allow comparing user event sequences with an optimal sequence. However, these tools rarely provide suggestions to help designers improve their artifacts; the expert is still needed to detect concrete usability problems in the deviations among event sequences, and find a solution (Fernandez et al., 2011). Moreover, the set of usability problems that can be recognized by comparing event sequences is limited (for instance, a frequently performed activity could be unnecessarily long for all users). In turn, there are many sources of usability guidelines and good practices in the literature, though it is still hard for a developer to identify which of these guidelines address a particular problem that appears on a running application.

Our proposal for overcoming these two issues related to dynamic usability assessment and repair is to extend event logging analysis to report concrete problems that are solvable through refactoring. Being an agile practice, refactoring allows improving usability in an incremental way using feedback from users, even (especially) in already deployed applications (Garrido et al., 2011). Moreover, refactorings are beneficial not only as cataloged, mechanized solutions, but also because each solution is linked to the particular problem or "smell" that it solves. In the case of usability, we call them *usability smells* (Garrido

* Corresponding author.
E-mail addresses: julian.grigera@lifia.info.unlp.edu.ar (J. Grigera), garrido@lifia.info.unlp.edu.ar (A. Garrido), mrivero@lifia.info.unlp.edu.ar (J.M. Rivero), gustavo@lifia.info.unlp.edu.ar (G. Rossi).

http://dx.doi.org/10.1016/j.ijhcs.2016.09.009
Received 12 December 2015; Received in revised form 13 September 2016; Accepted 17 September 2016
Available online 19 September 2016
1071-5819/ © 2016 Elsevier Ltd. All rights reserved.

et al., 2011).

In our current work, we aim at providing automatic advice about usability smells of user interaction for deployed web applications. Our automated strategy to usability smell recognition is based on the analysis of user interaction (UI) events. Thus, we extend previous work in this area by linking specific UI events to usability smells, defining new usability smells, and reporting usability smells on-the-fly at an abstraction level which makes it possible to suggest concrete solutions for them in terms of refactorings.

We have implemented the approach in a tool called *USF* (*Usability Smells Finder*). The tool can be used as a service (SaaS) with minimal setup effort, and is able to provide up-to-the-minute advice for deployed web applications. It is implemented in a way that allows for the extension of usability smells' detection strategies. Therefore, it is targeted to a broad audience of practitioners with different levels of usability expertize. On the one hand, usability experts may use USF to get rapid feedback of real interactions from a mass of users, configuring the tool to their needs. On the other hand, developers without usability expertize may use USF after a simple installation, let the tool gather evidence to diagnose usability problems and implement the solutions that it suggests.

The structure of the rest of the article is as follows: the next section presents background on usability refactoring and our catalog of usability smells for user interaction. Section 2 provides background and describes the differences and similarities with related work. Section 3 describes the usability smells used in this work, while Section 4 describes in detail the process of usability smell recognition and reporting, starting from the analysis of UI events, their abstraction to a mid-level concept called usability events, and the filtering and aggregation of usability events into usability smells. Section 5 provides the architecture and implementation details of the tool. Section 6 presents two experiments that we ran to assess the accuracy and versatility of the process and tool, and finally Section 7 concludes the article with contributions and future work.

2. Related work

In this section, we will review different approaches related to different aspects of our proposal. We first provide a background on refactoring and bad smells, their specific application in the context of web usability, and our previous work on the definition of usability smells. We then review usability evaluation methods in general, and place our work into their classifications. Then, we review log analysis and visualization methods, following with remote user testing and finally we describe the most relevant approaches we found to be closer to our proposal.

2.1. Background on usability and refactoring-based tools

Our work in usability takes many ideas from code refactoring. Refactoring was originally defined by Opdyke as a transformation that preserves behavior, aimed at improving the internal design of code (Opdyke, 1992). Later on, Fowler popularized the technique by publishing a catalog of refactorings for object-oriented code (Fowler, 1999). The refactorings in Fowler's catalog seek to improve internal quality measures like understandability, extensibility and maintainability of the different components in an object-oriented program like methods, classes and hierarchies, as well as data and conditional expressions. For example, "Extract Method" turns a piece of code into its own method, with an appropriate name that explains its purpose. The power of refactoring lies in helping non-experts to identify potential problems in the target aspect and traverse through a series of small steps towards a good solution for those problems. In the refactoring jargon, these potential problems are called *"bad smells"*, and their presence likely means that the code needs refactoring. For example, the refactoring "Extract Method" is intended to solve smells like "Long Method" and "Duplicate Code" (Fowler, 1999).

The refactoring technique became an essential practice of agile methodologies, and its scope was soon extended to other programming paradigms and beyond improving internal qualities of code into improving external qualities of software, like database safety (Ambler and Sadalage, 2006), parallel programs' performance (Dig, 2011), and web application's navigability (Cabot and Gómez, 2008). In 2007, we started working on the application of refactoring to improve the usability of web applications (Garrido et al., 2007). We defined *usability refactorings* as changes to the navigation, presentation or business processes of a web application with the purpose of improving its usability, while preserving the expected functionality and result (Garrido et al., 2011). With these refactorings, developers may attain usability enhancements like a balanced distribution of content in the screen and among pages, a better navigation structure and process workflow, proper support for the user while executing a business process, etc. An example of a usability refactoring is *"Provide Breadcrumbs"*, to help users keep track of their navigation path up to the current page (Garrido et al., 2011).

Similarly to bad smells in code ("code smells" for short), we have defined *usability smells* as indicators of possible problems that need refactoring (Garrido et al., 2011), where the problems relate to any aspect of the quality in use of a web application: effectiveness in use, efficiency in use, or satisfaction in use (ISO, 2011). In our earlier work, usability smells were cataloged to be manually recognized at different model levels (Garrido et al., 2011). Examples of these smells are "Absence of meaningful links" (in the navigation model), "Cluttered interface" (in the presentation model), and "Long activity" (in the process model).

In a later work, we developed a framework that allows applying usability refactorings on the client side, thus reducing the cost of changing a running system at the server (Garrido et al., 2013). We also developed a catalog with new usability refactorings and usability smells (Distante et al., 2014). The refactorings in this catalog may be applied either at the model level, in a model-driven development approach or at the client-side, and usability smells may be discovered in the models or by manually inspecting the results of user tests. An example of usability refactoring from that catalog is *Change the widget used to execute an activity*, aimed at replacing a widget that has been found awkward to use or produce errors for a more appropriate one, e.g. replacing free textboxes with calendar widgets for selecting dates, or selection boxes for ranged values. In the catalog, the usability smell that triggers this refactoring is *Risk of error*. Other cataloged usability smells are *Difficult access to information, User confusion*, and *Process inflexibility* (Distante et al., 2014).

Moreover, we have conducted a statistical test to measure the real gain that usability refactorings produce on effectiveness in use, efficiency in use and satisfaction in use (Grigera et al., 2016). For the experiment, detecting usability smells was not a simple task, as it required manual expert intervention to go through the results of user tests.

Finally, in the context of refactoring, our process for smells detection could be compared to the work of Lanza and Marinescu (2006) to systematically detect bad smells in code: they use well-known object-oriented metrics and metric-based patterns as detection strategies to identify potential bad smells and structural design problems and provide the appropriate refactorings as recovery means. In a similar way, we use UI event patterns to characterize usability smells and suggest usability refactorings for most cases.

2.2. Usability evaluation methods

Like Hornbæk (2006) said in his review back, usability cannot be directly measured, so researchers must select usability aspects that can be measured and represent valid indicators of usability, like in the model proposed by Seffah et al. (2006). There are different ways to