A Tool for Detecting Bad Usability Smells in an Automatic Way

Julián Grigera[1], Alejandra Garrido[1,2], and José Matías Rivero[1,2]

[1] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, Argentina
[2] CONICET, Argentina
{Julian.Grigera,Garrido,MRivero}@lifia.info.unlp.edu.ar

Abstract. The refactoring technique helps developers to improve not only source code quality, but also other aspects like usability. The problems refactoring helps to solve in the specific field of web usability are considered to be issues that make common tasks complicated for end users. Finding such problems, known in the jargon as *bad smells*, is often challenging for developers, especially for those who do not have experience in usability. In an attempt to leverage this task, we introduce a tool that automatically finds bad usability smells in web applications. Since bad smells are catalogued in the literature together with their suggested refactorings, it is easier for developers to find appropriate solutions.

1 Introduction

The refactoring technique [1] has been recently brought to the usability field of web applications, allowing developers to apply usability improvements without altering the application's functionality [2]. The problems developers can solve by refactoring, called *bad usability smells* are often hard to find, so there are ways to assist this task.

Running usability tests is the most common way to find usability problems [3], but it requires supervision by usability experts, among other resources. These tests can be automated to some extent, presenting an attractive alternative to lower the costs. Some approaches in the literature automate the gathering of data from users [4, 5] but not the analysis, which still depends on usability experts. Other approaches automate part of the analysis by comparing users behavior to optimal behavior paths [6, 7], but this requires prior preparation and subjects to conduct the experiments. There are also commercial tools like CrazyEgg[1] or ClickTale[2] that offer statistical data to their customers by analyzing interaction data from real users instead of tests subjects. However, even if these tools can represent a cheaper option, the results they obtain also require analysis from usability experts.

The tool we present in this work also automates the gathering of interaction data from real users, but in addition, it preprocesses the events on the client side to report concrete usability problems, easier to interpret than mere statistics. Moreover, the tool presents the usability issues as *bad usability smells*, which are problems catalogued in the literature along with the *refactorings* that solve them. Using these catalogues, developers can find a

[1] http://www.crazyegg.com
[2] http://www.clicktale.com

concrete way to correct the detected bad smells. The *Bad Smells Finder* (as we called our tool) was developed as the first stage of a process for automatically improving usability on web applications. We explain this process in the next section.

2 The Process in a Nutshell

Our process for improving web usability is based on the refactoring technique. When applying refactoring in the context of web usability, developers first must detect bad usability smells, and then they must find refactorings to solve them, keeping the basic functionality intact. The tool helps them find bad usability smells.

The automated process for finding bad smells consists in three steps, depicted in Fig. 1. The **Threats Logger** is a client-sided script that gathers interaction events from real users. Instead of logging raw, atomic events, it processes them to generate *usability threats*, a concept we devised to represent higher-level interaction events.

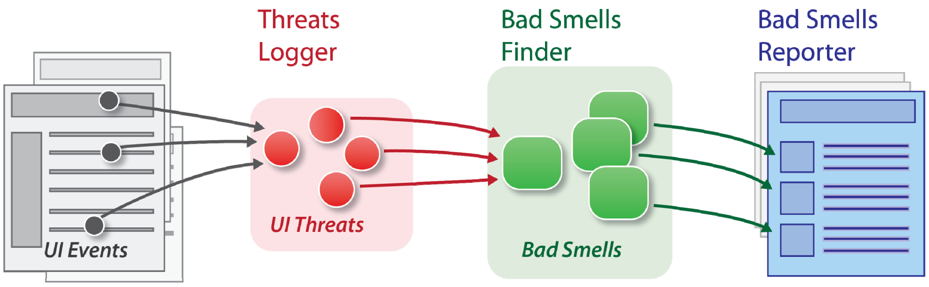


Fig. 1. Schematics of the process

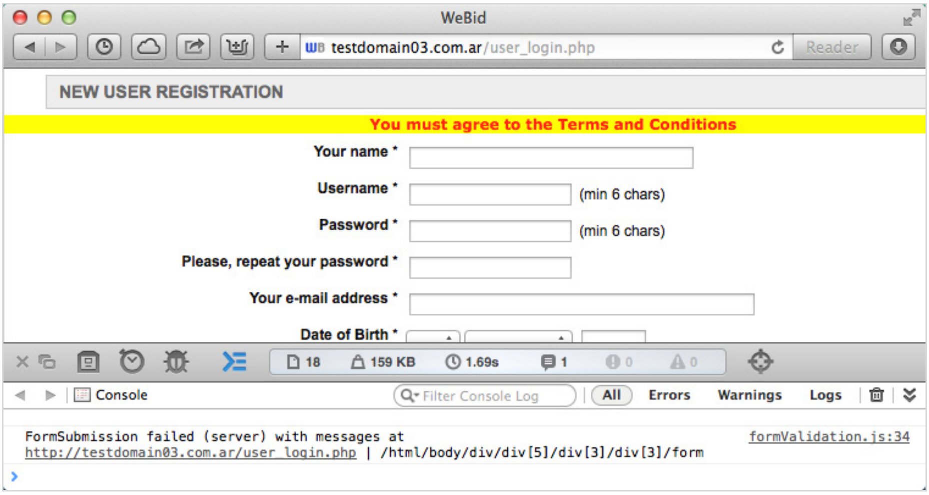
The server-sided **Bad Smells Finder** receives usability threats and stores them for analysis. When a user requests a report, the Bad Smells Finder processes the threats and displays the resulting bad smells through the **Bad Smells Reporter** frontend.

3 The Tool in Action

We will show how the tool works with an example. Consider for instance a web application where users need to register before they can operate. Whenever a user fills the registration form, the threats logger captures the form submission event and evaluates what happens next:

- If no navigation follows, the threats logger considers the submission was blocked by **client-side validation**.
- If a navigation is detected, and the form is still on the destination page, the logger considers there was **server-side validation**.
- If a navigation is detected, but the form is absent in the destination page, the logger considers a **successful submission**.

The tool uses a simple algorithm to identify search forms, where validation rules do not generally apply. Combining all this information on the client-side, it creates a *Form Submission* threat and sends it to the Bad Smells Finder. The script can be set to *verbose* mode to show the threats it finds in the browser's console, as seen in Fig. 2.



NEW USER REGISTRATION

You must agree to the Terms and Conditions

Your name *		
Username *		(min 6 chars)
Password *		(min 6 chars)
Please, repeat your password *		
Your e-mail address *		
Date of Birth *		

```
FormSubmission failed (server) with messages at                    formValidation.js:34
http://testdomain03.com.ar/user_login.php | /html/body/div/div[5]/div[3]/div[3]/form
```

Fig. 2. Threats Logger indicating the detection of a *Failed Submission* threat

The server-side Bad Smells Finder processes all Form Submission threats to potentially find a *No Client Validation* bad smell, which indicates a problematic form that usually fails to submit without offering any client-side validation whatsoever. To do this, it compares the amount of successful submissions with the ones that failed with server validation, according specific criteria for the proportion threshold (e.g. 30% of failed validations indicate a bad smell).

The site owners may then ask for a report by accessing the tool's Reporter, where bad smells are listed with data like the URL where it happened, an XPath of the affected element, and specific extra information depending on each bad smell.

The Bad Smells Finder can detect 12 different kinds of bad usability smells, and the logics for detecting each one are diverse. Other featured bad smells are:

- **No Processing Page:** By calculating the average time of a request and watching DOM mutations, the Bad Smells Finder is able to detect that a process usually takes a long time, but users are never informed about that process taking place on the background (i.e. "loading…" widget).
- **Unnecessary Bulk Action:** Users perform actions on a list of items by first marking checkboxes, and then selecting the action – e.g. deleting emails on a webmail application. If the Bad Smells Finder detects that most of the time users apply actions one item at a time rather than many, then the Unnecessary Bulk Action is detected, implying that the UI with checkboxes and actions should be complemented with other mechanism that require less interactions.
- **Free Input For Limited Values:** A free text input is presented to the user, but the set of possible values that can be entered belong to a limited set, like countries or occupations. Two problems ensue: error-proneness, and unnecessarily time wasted in typing the whole text. The Bad Smells Finder captures all the inputs and calculates the proportion of repeated (and similar) values, in order to determine the bad smell's presence.

The rest of the bad smells are related to navigation issues (like long paths for frequently accessed pages), and misleading/misused widgets. We are currently extending the tool to detect more bad usability smells.

4 Tool Implementation and Usage

The tool has two main modules: the Threats Logger, implemented as a client-side script, and the Bad Smells Finder, a server-sided component that analyzes threats and reports bad smells.

The client-side Threats Logger (coded in JavaScript using the JQuery[3] library) captures interaction events and then processes them on the client to create (and filter) usability threats. The server-sided analyzer parses the incoming asynchronous POST requests from the client script to generate usability threats. When a report is asked, the analyzer filters all the threats to find potential bad usability smells, and then renders the bad smells report in the web frontend.

To install the tool, the site owner must include the Threats Logger script in the application's header. After completing this step, the Bad Smells Finder starts logging and reporting bad usability smells right away.

References

1. Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D.: Refactoring: Improving the Design of Existing Code. Object Technology Series. Addison Wesley (1999)
2. Garrido, A., Rossi, G., Distante, D.: Refactoring for Usability in Web Applications. IEEE Softw. 28, 60–67 (2011)
3. Rubin, J., Chisnell, D.: Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests. Wiley (2008)
4. Atterer, R., Wnuk, M., Schmidt, A.: Knowing the user's every move. In: Proceedings of the 15th International Conference on World Wide Web, WWW 2006, p. 203. ACM Press, New York (2006)
5. Saadawi, G.M., Legowski, E., Medvedeva, O., Chavan, G., Crowley, R.S.: A Method for Automated Detection of Usability Problems from Client User Interface Events AMIA 2005 Symposium Proceedings, pp. 654–658 (2005)
6. Fujioka, R., Tanimoto, R., Kawai, Y., Okada, H.: Tool for detecting webpage usability problems from mouse click coordinate logs. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 438–445. Springer, Heidelberg (2007)
7. Okada, H., Fujioka, R.: Automated Methods for Webpage Usability & Accessibility Evaluations. In: Adv. Hum. Comput. Interact, ch. 21, pp. 351–364. In-Tech Publ. (2008)

[3] `http://jquery.com`