Kobold: Web Usability as a Service

Julián Grigera[1], Alejandra Garrido[2], Gustavo Rossi[2]
LIFIA, Universidad Nacional de La Plata, Argentina
[1]Also at CIC, Argentina
[2]Also at CONICET, Argentina
{julian.grigera, garrido, gustavo}@lifia.info.unlp.edu.ar

***Abstract*—While Web applications have become pervasive in today's business, social interaction and information exchange, their usability is often deficient, even being a key factor for a website success. Usability problems repeat across websites, and many of them have been catalogued, but usability evaluation and repair still remains expensive. There are efforts from both the academy and industry to automate usability testing or to provide automatic statistics, but they rarely offer concrete solutions. These solutions appear as guidelines or patterns that developers can follow manually. This paper presents Kobold, a tool that detects usability problems from real user interaction (UI) events and repairs them automatically when possible, at least suggesting concrete solutions. By using the refactoring technique and its associated concept of bad smell, Kobold mines UI events to detect usability smells and applies usability refactorings on the client to correct them. The purpose of Kobold is to deliver usability advice and solutions as a service (SaaS) for developers, allowing them to respond to feedback of the real use of their applications and improve usability incrementally, even when there are no usability experts on the team. Kobold is available at: http://autorefactoring.lifia.info.unlp.edu.ar. A screencast is available at https://youtu.be/c-myYPMUh0Q**

***Index Terms*—**Web Usability, Software as a Service, Usability Refactoring

I. Introduction

Usability problems affect a great number of web applications. While companies recognized that usability is crucial to stand a chance facing the competition, the process of evaluating and improving usability remains too expensive [1], [2]. A common approach to evaluate usability is by conducting user tests, which has the advantage of capturing real usage data [3]. However, they require considerable time and resources to recruit users and hire experts to design the tests, analyze the results, discover problems and find solutions for them. Hence, larger companies are mostly the ones that may afford it. There are efforts to minimize the cost of user testing with automated tools that log user interaction (UI) events, perform log analysis and in some cases provide sophisticated visualization of UI events that hint usability flaws [2]. However, an expert is still required to find a good solution to each problem, since guidelines available in the literature are still hard to relate to a particular problem that appears on a running application.

We are motivated by the need to provide tools to make usability improvement affordable, i.e., simple and inexpensive, even to non-usability experts. Our goal is to provide automated advice for developers on the usability problems that users encounter while interacting with their website and the possible solutions, even applying them when possible. Thus, our tool allows taking heed of customer feedback. Moreover, the approach is compatible with an agile development process, where usability problems can be detected during the cycle after a release, and solutions can be applied by the team in the next cycle.

In agile methods, refactoring is an essential technique for incremental improvement. It allows applying changes in small steps after identifying potential problems in the code, called "bad smells". We have proposed refactoring for the incremental improvement of web usability, and defined **usability refactorings** as changes to navigation, presentation or business processes of web applications with the aim of improving usability, while preserving the functionality and result [4]. In a similar way of code smells, we defined **usability smells** as indicators of possible problems related to *usability in use*. These problems encompass any issue with user interaction that makes the completion of tasks difficult or confusing. For example, the usability smell *No Client Validation* indicates that a form performs data validation on the server, forcing users to wait for the request to complete.

We have implemented the approach for automatic usability improvement in a tool called Kobold (in reference to the invisible sprites that perform domestic chores in the Germanic folklore). In essence, the contributions of Kobold are:

- it allows inspecting feedback of the real use of a web application regarding UI events;
- it reports usability smells of user interaction as soon as they are detected;
- it suggests solutions in terms of usability refactorings that when possible are self-installable in the client.

Different stakeholders can take advantadge of each stage of the process depending on their expertise on usability. Anyone from *novice users* to *usability experts* could make use of the automated detection of problems and refactoring. For instance, in an agile development team, reports of usability smells may be of interest to *team members* as well as *product owners*. Regarding refactorings, *developers* without usability expertise can apply automated ones with little effort. Moreover, non-automated refactoring suggestions provide enough detail so developers can code them manually. The *usability experts* in the team, even if they don't have programming skills, could analyze the events separately to define new usability smells, and design new refactorings for the developers.

978-1-5386-2684-9/17 © 2017 IEEE

ASE 2017, Urbana-Champaign, IL, USA
Tool Demonstrations

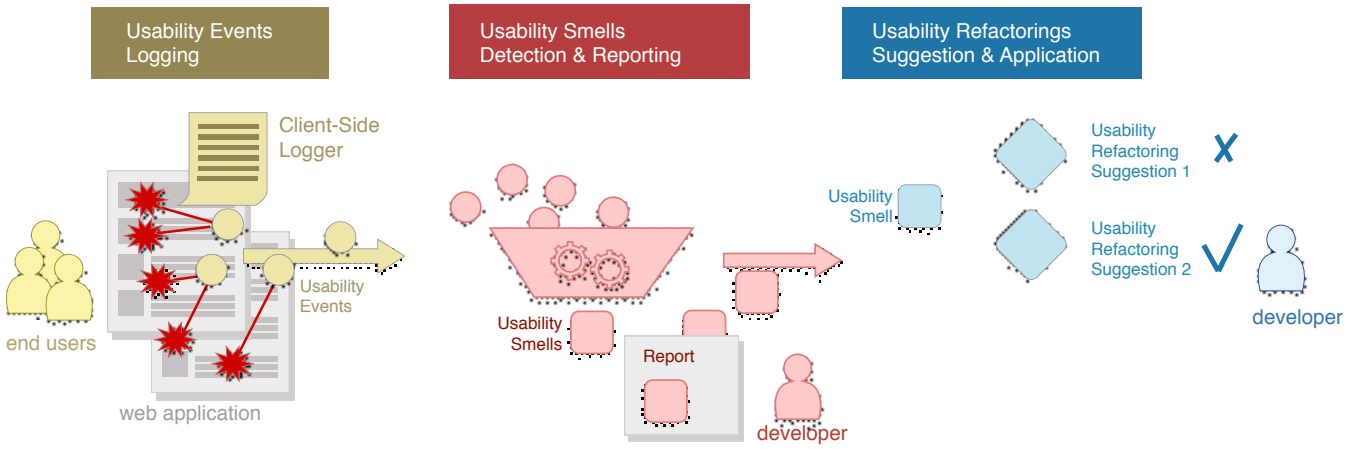

Figure 1: Steps of the automated approach for Web Usability improvement.

II. An Automated Approach for Web Usability

In order to assist developers to find and correct usability problems on their web applications, we devised a process centered in usability smells and refactorings. This was conceived to maximize automation, requiring the minimum possible effort from developers. To achieve this, the process relies on end-user's interaction events as the main input.

There are 3 stages to the process. The first stage consists in mining UI events in the client and collecting relevant ones. In the second stage, usability events are analyzed in an off-site server to detect usability smells. The third and last stage consists in recommending usability refactorings to solve the detected smells. In some cases, these refactorings can be applied automatically or semi-automatically with some input from developers. The first two stages were already implemented in a tool called Usability Smells Finder (USF) [5], and this paper describes the implementation of the last step, hence completing the tool support for the full approach. An overview of the process can be seen in Figure 1.

The first stage consists in mining UI events from users and aggregating them in *usability events*. These events are customized to capture information that's relevant to potential usability problems (i.e. smells), and also to prevent overloading the server with irrelevant information. This is achieved by doing client-side pre-processing of low-level events before sending the information to the server. There are over 16 kinds of usability events [5]. An example of usability event is *Search*, which represents a search made by a user, capturing the time, search term, the search form's HTML code and location within the page, whether any results that include the search term were found, and also whether a link within those results was clicked (i.e. results were potentially useful).

Stage two of the process is where usability smells are detected by processing the usability events. For each kind of smell, there is a separate component analyzing the incoming events. For example, to detect the *Scarce Search Results* smell, that indicates that a search form is not bringing results to the users, there is a component that analyzes the Search events and calculates the proportion of successful vs. unsuccessful searches. Currently USF is able to detect 16 kinds of usability smells.

The third stage is where refactorings are suggested and applied. Refactoring suggestion is based on existing catalogs that relate each usability smell with one or more refactoring [4]. However, in these previous catalogs, refactorings were usually defined at a design level and in abstract terms, so we had to refine them to be able to build more concrete, self-installable refactorings. The generation of refactorings is a complex task, but the detailed diagnoses of smells helps considerably in their implementation. The next section describes this stage in detail.

III. Kobold: Usability as a Service

Kobold is the tool that implements the full process for automating web usability improvement. It is a web application itself, and works as a SaaS (Software as a Service) solution for web developers. The main features that the tool offers are:

- Automatic detection of potential usability issues stated as usability smells, with specific details.
- Suggestion of solutions for the detected usability smells in terms of usability refactorings.
- Automatic application of refactorings when possible.
- Instant report and browsing of usability events.

In the next subsections, we show how the developers can incorporate the use of the tool to improve their already deployed applications, then we provide some detail on the refactoring process, and finally we show some refactoring examples.

A. Using the Tool

To start using Kobold, developers must register an account, and the tool will provide them with a JavaScript snippet to embed in their application's header. With the snippet in place, the tool immediately begins to log end-users' interaction and to search for usability smells.

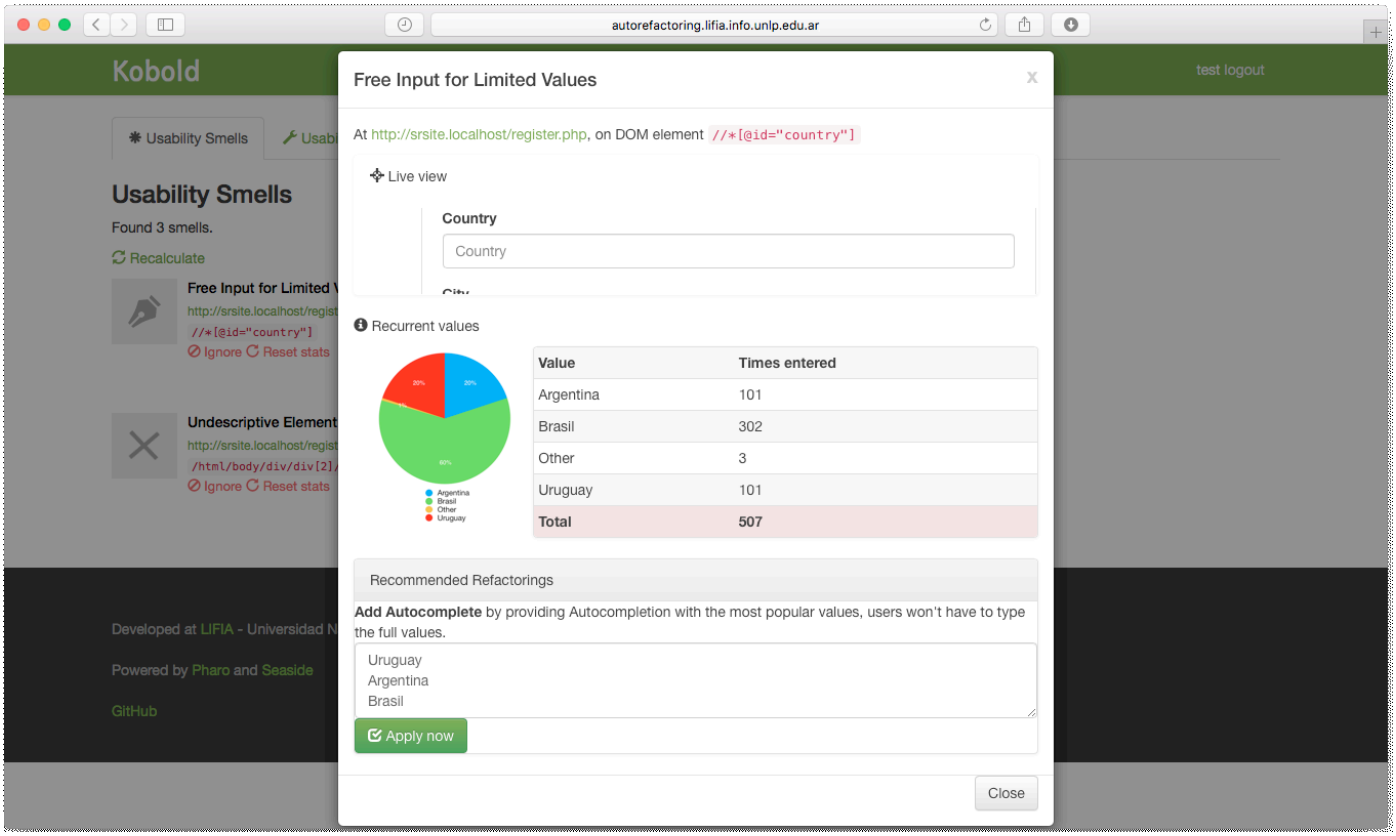

Figure 2: Kobold screenshot. In the background, the report on all detected usability smells, and in the pop-up window, specific information about the smell Free Input for Limited Values and the recommendation for the Add Autocomplete refactoring, ready to be applied.

When a usability smell is reported, Kobold suggests one or more refactorings to solve it. Depending on the case, the refactoring may be presented as a suggestion (to be implemented by the developer), or it can be offered as an automated solution. For some refactorings, the tool will need extra information from the developer before applying it - we call these *semi-automated* refactorings.

Notice that in order to use the tool, the user doesn't need to be an expert in usability, since the reports can be understood by anyone with a basic knowledge of web interfaces.

Figure 2 shows a screenshot of Kobold suggesting a refactoring called *Add Autocomplete* to correct the usability smell "Free Input for Limited Values". The screenshot shows the usability smell's diagnosis with a live view of the affected element (the "Country" input), and prompts the user to confirm the application of the refactoring, or edit the values for the autocomplete's suggestions.

B. Taxonomy of Client-Side Usability Refactorings

We have been able to match each usability smell of user interaction with one or more concrete usability refactoring to solve it. In consequence, Kobold can always suggest at least one solution for each usability smell that it automatically detects. In some cases, Kobold may even select the best solution among a few for a given smell. However, usability refactorings may not always be applied automatically at the client-side, for example in the case of *Split / Postpone activity*, which solves the smell "Abandoned Form". Thus, regarding the extent to which Kobold can automate refactorings, we have classified them in 3 groups:

1. **Suggestion Only:** these refactorings can only be suggested to be implemented by a developer.
2. **Semi-Automated:** they can be automatically generated, but require some parameter data that a developer must provide manually.
3. **Fully Automated:** the detected smell provides all data necessary to generate and apply the refactoring without any external help.

There is a second aspect in which we classify the refactorings that may be applied with some degree of automation (i.e. groups 2 and 3). This criterion refers to the target UI element where the refactoring must be applied. In some cases, the same smell may affect more than one element in a site, and Kobold has the ability to detect all elements affected by the same smell thanks to its specific similarity algorithm [5]. In this aspect, we contemplate 2 possibilities:

a) **Single Element**: the refactoring is applied on a unique element affected by a usability smell. For example, a heading, or a login form.

b) **Elements Family**: the refactoring must be applied on a group of similar elements affected by the same smell. For example, all products included in a search results page, or a list of news.

In the next subsection we exemplify with some refactorings from all these categories.

C. Examples of Usability Refactorings

1) Turn Attribute into Link (Group 2; Target a / b): This is a simple refactoring that converts a static UI element, such as a text or image, into a link. The usability smell that this refactoring solves is called *Unresponsive Element*, which indicates that an element is frequently being clicked, but produces no action. An example of this symptom is a heading logo that does not lead users to the website's homepage when clicked.

Regarding the degree of automation, the *Turn Attribute into Link* refactoring is **semi-automated**, since it requires end users to provide the link destination, whereas the rest of the implementation is generated automatically thanks to the detailed information that USF reports with each smell. With respect to the target, it can be applied either to a single element, or to an elements' family. In the last case, Kobold can use an expression generated by the similarity algorithm during smell detection to refactor all related DOM elements belonging to a same family at once.

2) Add Autocomplete (Group 2 / 3; Target a): Sometimes forms include text fields with free input for data that belongs to a limited set of values, either a closed set (like countries) or an open one, but with a dominant amount of popular choices (like occupations). The usability smell *Free Input for Limited Values* detects these situations. It does so by analyzing the entered values and comparing similitude amongst them. When it detects that most values are within a closed set, with a small percentage of outliers, the smell is signaled. Some input types like credit cards or passwords are discarded to preserve privacy. Also, personal data like names or addresses, rarely trigger this smell because of their inherent diversity.

When *Free Input for Limited Values* appears, Kobold suggests three possible refactorings: *Replace Widget*, *Add Default Value* or *Add Autocomplete*. This suggestion is based on the variety of values that the smell detects. If all values belong to a closed set of up to 8 values, and the outliers' percentage is marginal, Kobold suggests the use of a radio buttons list via a *Replace Widget*. If the values are sparser but still within well-defined sets (e.g. cities names, even misspelled ones) then an *Add Autocomplete* is suggested. If there is a dominant value that outnumbers all others, *Add Default Value* is suggested.

There is another use for *Add Autocomplete*. The smell *Scarce Search Results* indicates that a search form usually fails to bring results. In these cases, Kobold can also suggest this refactoring but offering the popular values from *successful* searches as term suggestions.

The refactoring *Add Autocomplete* applies to a single text input field, taking the values that USF found as popular for the field, and providing them as suggestions when users are completing it. This refactoring can be used both as **fully automated** or **semi-automated**, since the user has the ability of correcting or providing new values to the list of suggestions that will drop down in the autocompletion (e.g. by adding a city that nobody entered).

3) Distribute Menu (Group 1; Target b): The usability smell *Forced Bulk Action* detects a common interaction issue with lists of selectable items. With the intention of easing the application of an action to many items at once, a usual design consists in providing checkboxes to select multiple items, and buttons to apply actions to all of them (e.g. delete). While this interaction style is effective for this scenario, it becomes a usability smell when most users apply the action on a single item. As a solution for this smell, the *Distribute Menu* refactoring proposes adding a button for each popular action to every item on the list (see Figure 3).

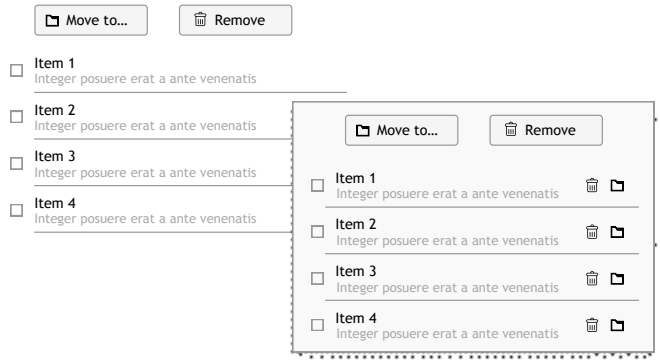

Figure 3. Distribute Menu refactoring. The top-left figure shows the page before refactoring. The refactored, bottom-right version includes a "move" and "remove" button for every single item.

Due to the complexity of detecting how the actions are performed (and, consequently, determining how to reproduce them), at this time we have not found a way to automatically or semi automatically apply this refactoring, so it can only be suggested (**suggestion only**). Nevertheless, Kobold is able to assist the developer by providing the XPath locator for the family of affected items (i.e. the items that comprise the list). This refactoring applies over an **element's family**.

From a total of 15 refactorings that Kobold currently suggests, 12 of them can be fully or partially automated. Besides *1)* and *2)* above, other refactorings currently implemented in Kobold include: *Add Processing Page* ("loading…" overlay for slow processes), *Add Default Value* for text inputs, *Add Validation* (fully automated) that incorporates inline validation for frequently rejecting forms. There are also different *Replace Widget* refactorings, e.g. for incorporating date pickers, combo boxes or radio buttons.

IV. Tool Arquitecture

Kobold was built on top of a previous tool called Usability Smells Finder (USF) [5]. USF captures UI events from real users at the client-side, mines and filters the events on-the-fly, and aggregates and classifies the events at the server-side using specialized algorithms that discover usability smells. To be able to log usability events from end-users, USF provides a client-side JavaScript snippet that developers must embed in their applications. Given the immediate nature of the logs analysis, the

tool reports the usability smells that may affect the web application right as they appear. The imported library is very lightweight (~30kb minified) and the mining, being asynchronous, is unnoticeable by the end-users, as also confirmed by our experiments in real world web sites.

Kobold adds to USF the capability of suggesting and even applying usability refactorings. This is possible also thanks to the JavaScript client-side component. Once loaded, besides starting to log usability events, the component asks the Kobold server for potential refactorings. If the server finds any refactoring that must be applied, it generates them and sends the JavaScript code to be executed. The technology for applying refactorings uses a client-side adaptation framework that adapts existing applications on the client-side by changing their DOM [6]. In the case of Kobold, these adaptations were refined and restructured to fit the refactorings in our catalog.

When refactorings are semi or fully automated, Kobold generates the JavaScript code from a fixed template. If the refactoring is applied to a DOM element, the template is dynamically completed with the XPath locator for it. In semi-automated refactorings, the user input is also added, like extra terms for autocompletion. If any libraries are required (e.g. a Date Picker plugin) they are imported once, but there is one instantiation code for each refactored element. The full code (i.e. libraries importing and instantiation for each element) is sent from the server at page loading if the elements affected by usability smells are present. The code generation is designed to be easily extended in new refactorings by providing hotspots for incorporating libraries, automated variable declaration for the affected elements, and other features.

Using the aforementioned techniques, Kobold can offer usability refactorings application, either automatically or semi-automatically. Note that each sub-component of Kobold is engineered as a framework, so experts may extend it with new usability events, smells, or refactorings.

V. VALIDATION

In this section, we briefly describe the early validations on usability smells detection, and the first results on a new experiment in which we assess the usability refactoring application.

A. Usability Smells Validation

In our previous work on usability smells detection, we presented an experiment where we validated aspects of the first two stages of the process, namely **usability events logging** and **usability smells detection** [5].

For the first part, we validated that the heuristics for detecting end user events were correct. Some heuristics involve interpreting users' intention, and others involve interpreting the role of the affected interface elements.

To assess all different heuristics, we ran a Precision & Recall analysis comparing the tool findings with a direct observation in a controlled environment on 3 websites with 15 volunteer users (8 m. and 7 f., ages from 26 to 64 $\bar{x}$ 35.66, s^2 75.52). We used a F2 score, which is a weighted average of precision and recall that puts more weight on the latter. We favored recall for the event detection experiment since it is more important not to "miss" events that cannot be recovered. False positives, while undesired, can be dealt with in the smells detection stage. Results showed that, in the case of heuristics involving user intention (5 event kinds), the tool performed with an F2 score between .75 and .97. In the heuristics involving reasoning on GUI elements (3 events), the F2 score was between .67 and .75. The rest of the heuristics (for 4 events) had no margin for error and thus were not assessed.

The second part of the experiment consisted in evaluating the usefulness of the usability smells reporting. To do this, we compared the usability problems found in a user test with 9 volunteers (4 m. and 5 f., ages from 25 to 62, $\bar{x}$ 35, s^2 125.25). In this case, we ran a GQM (Goal-Questions-Metrics) experiment with 6 different metrics. The most relevant metric was the percentage of problems found by USF with respect to traditional usability tests. At that time, the USF tool was able to find 52% of usability smells manually found in the control applications. Other metrics related to the reliability of the findings were true and false positives, 75% and 25% respectively.

B. Usability Refactoring Validation

We conducted another validation to assess the tool's ability to automatically improve usability on web applications. The validation consisted in running user tests on two real-world web applications: a merchandising online shop ("FF Site") and an online travel agency ("TA Site"). We had Kobold solve the smells detected by USF applying automated and semi-automated refactorings and compared the sites' usability with and without such refactorings. The refactorings applied were: *Add Autocomplete*, *Add Validation*, *Turn Attribute into Link*, *Add Default Value* and *Add Processing Page*.

We recruited 8 subjects, 5 males, 3 females (ages: $\bar{x}$ 37.5, s 9.74). The subjects ran tasks on both versions of the 2 applications, as follows:

FF Site

1. Find the price of a specific product.
2. Add it to the cart and go to the homepage.
3. Register.
4. Search for a given product.
5. Complete the checkout.

TA Site

1. Search for a given package.
2. Reserve the cheapest.
3. Search for a flight ticket.

We measured three aspects related to usability according the ISO/IEC 25010 standard: effectiveness as completion percentage, efficiency in average time, and satisfaction, by using a standard SUS questionnaire [7]. To avoid the learning effect threat, each subject interacted with one version of each application. This way, we gathered 8 results for each site: 4 for the refactored version and 4 for the original one.

The results showed improvements in all 3 measured aspects, which implies that Kobold is indeed able to improve usability automatically. Detailed results be seen in Table 1.

Table 1. Experiment's Results

	FF Original	FF Refactored	TA Original	TA Refactored
Satisfaction	74.375	**78.75**	84.375	**93.75**
Efficiency	03'40''	**02'57''**	03'09''	**02'36''**
Effectiveness	100%	**100%**	83.33%	**100%**

Although we still must extend the experiment with more refactorings and subjects, the results are very promising, based on Nielsen's rule that a single test with 5 subjects is able to detect nearly 85% of the total usability problems on a website[1].

VI. Related Work

The idea of analyzing user behavior in websites through UI events has become quite common, from web analytics tools (like Google Analytics, ClickTale, etc.) which mostly target improving revenue, to other tools like in our case that try to hint usability problems. Among the latter, there are tools like WUP [8] or WELFIT [9] that log events during remote user testing, and discover problems in the deviations with optimal logs from usability experts. Speicher et al. [10] developed tools for obtaining usability scores in Search Engine Results Pages (SERPs). Despite being focused only on SERPs, the approach is similar to ours, and it also provides solution's suggestions. While other approaches may succeed in pointing out usability problems, most of them require an expert opinion to interpret the outcomes and provide solutions, manually searching through usability guidelines in the literature [11], [12]. The tool that is closest to ours in terms of functionality is, to our knowledge, W3Touch [13]. The tool detects problems related to missed links and zoom levels on mobile devices by analyzing user logs. It also offers simple fixes that may be adjusted by developers, though the main difference is that W3Touch focuses on responsiveness issues for touch-based devices.

VII. Conclusions And Future Work

In this paper, we presented Kobold, a tool that is able to automatically suggest and, in some cases, apply usability refactorings that have already been established to be successful at improving web usability.

We showed that automated usability improvement is possible by running a validation on two real applications. We also planned a more comprehensive experiment to assess all the refactorings.

Future work includes determining if the accuracy in detection and reporting of USF is good enough to automate other refactorings still to be implemented. Another challenge of the current work is selecting the most appropriate refactoring, especially in cases where different refactorings may solve the same smell. Moreover, once Kobold has detected many usability smells, they could be prioritised. Kobold shows in first place the smells that most users run into, but other criteria for prioritizing them may be researched.

We are also planning a validation to assess the level of adoption that Kobold may have among developers. The experiment will be conducted in two phases: first, we will collect information on how developers use Kobold in a period of 1 month, and then they will answer a survey about the tool.

Finally, we are studying the process that follows the refactoring application: how can we automatically measure the alleged improvement that refactorings bring? Since there are no accepted standards for these metrics, we are researching on the best ones for measuring aspects of the applied refactorings.

[1] http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/

Acknowledgments

The authors acknowledge the support from the Argentinian National Agency for Scientific and Technical Promotion (ANPCyT), grant number PICT-2015-3000.

References

[1] J. Nielsen and H. Loranger, *Prioritizing Web Usability*. Pearson Education, 2006.

[2] A. Fernandez, E. Insfran, and S. Abrahão, "Usability evaluation methods for the web: A systematic mapping study," *Inf. Softw. Technol.*, vol. 53, no. 8, pp. 789–817, 2011.

[3] J. Rubin and D. Chisnell, *Handbook of Usability Testing: Howto Plan, Design, and Conduct Effective Tests*. Wiley, 2008.

[4] D. Distante, A. Garrido, J. Camelier-Carvajal, R. Giandini, and G. Rossi, "Business processes refactoring to improve usability in E-commerce applications," *Electron. Commer. Res.*, vol. 14, no. 4, pp. 497–529, Sep. 2014.

[5] J. Grigera, A. Garrido, J. M. Rivero, and G. Rossi, "Automatic detection of usability smells in web applications," *Int. J. Hum. Comput. Stud.*, vol. 97, pp. 129–148, 2017.

[6] S. Firmenich, G. Rossi, S. Gordillo, and M. Winckler, "A crowdsourced approach for concern-sensitive integration of information across the web," *J. Web Eng.*, vol. 10, no. 4, pp. 289–315, 2011.

[7] J. Brooke, "SUS - A quick and dirty usability scale," *Usability Eval. Ind.*, vol. 189, no. 194, pp. 4–7, 1996.

[8] F. Paternò, A. G. Schiavone, and P. Pitardi, "Timelines for Mobile Web Usability Evaluation," in *Proc. Int. Working Conference on Advanced Visual Interfaces - AVI '16*, 2016, pp. 88–91.

[9] V. F. de Santana and M. C. C. Baranauskas, "WELFIT: A remote evaluation tool for identifying Web usage patterns through client-side logging," *Int. J. Hum. Comput. Stud.*, vol. 76, pp. 40–49, 2015.

[10] M. Speicher, A. Both, and M. Gaedke, "S.O.S.: Does Your Search Engine Results Page (SERP) Need Help?," in *Proc. ACM Conf. on Human Factors in Comp Systems - CHI '15*, 2015, pp. 1005–1014.

[11] L. Wroblewski, "Web Form Design: Filling in the Blanks," *Interactions*, vol. 0, no. October, p. 226, 2008.

[12] C. Mariage, J. Vanderdonckt, and C. Pribeanu, "State of the Art of Web Usability Guidelines," *Handb. Hum. FACTORS WEB Des.*, pp. 688--700, 2004.

[13] M. Nebeling, M. Speicher, and M. Norrie, "W3touch: Metrics-based Web Page Adaptation for Touch," *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. - CHI '13*, p. 2311, 2013.