Usage-Based Automatic Detection of Usability Smells

Patrick Harms and Jens Grabowski

Institute of Computer Science, University of Göttingen,
Göttingen, Germany
{harms,grabowski}@cs.uni-goettingen.de
http://swe.informatik.uni-goettingen.de

Abstract. With an increasing number of supported devices, usability evaluation of websites becomes a laborious task. Therefore, usability evaluation should be automated as far as possible. In this paper, we present a summative method for automated usability evaluation of websites. The approach records user actions and transforms them into task trees. The task trees are then checked for usability smells to identify potential usability issues. The approach was applied in two case studies and shows promising results in the identification of four types of usability smells.

Keywords: task trees, usage-based, automatic usability evaluation.

1 Introduction

Usability engineering is important to develop high quality websites [1]. But a fully-fledged application of traditional usability evaluation methods like user-oriented usability tests applied for several devices for displaying websites requires a high effort [2]. Therefore, in the recent years usability evaluation strives for automation and, hence, cost reduction.

In this paper, we present a new method for automated detection of indicators for usability issues, so called usability smells, based on recorded user actions on websites. The user actions are first transformed into task trees using a methodology proposed by us in [3]. Then, we scan the task trees for patterns of usability smells. As a result, our approach generates a list of detected usability smells. For each smell, it provides details about the user tasks and actions that are related to the smell, as well as the elements of the websites (e.g., links and buttons) that are involved. Our method detects four different types of usability smells. We show in this paper, how the smells are detected and clarify if our approach can be used to automatically detect usability issues.

The paper is structured as follows. First, we introduce the terminology and basic concepts used throughout the paper in Section 2. Then, we describe the detection of usability smells using the generated task trees in Section 3. Afterwards, we depict the validation of our approach in two case studies in Section 4 and discuss the respective results in Section 5. Finally, we compare our approach with related work in Section 6 and conclude on our findings with potential for future work in Section 7.

S. Sauer et al. (Eds.): HCSE 2014, LNCS 8742, pp. 217–234, 2014.

2 Background

Usability is the result of executing tasks with *effectiveness*, *efficiency*, and *satisfaction* [4]. It is influenced by the *usage context* which comprises the tasks to be executed, the physical environment, and the users performing the tasks. In this paper we consider the usability of websites and, hence, users of websites and the tasks they execute.

A *usability issue* is a problem that occurs during the usage of a website and decreases one or more of the factors effectiveness, efficiency, and satisfaction for a specific usage context. The problems can be caused by the design of the website, the functionality, the performance, and other aspects. As an example, a login process of a website can be inefficient (usability issue) because users have a long navigation to the login mask (design).

As *usability smell*, we refer to exceptional user behavior that may indicate a usability issues. For example, if users typically require many actions to perform a login on a website (usability smell) this may indicate a usability issue with respect to inefficiency of the login process. A usability smell has a description and refers to potential usability issues reflected by the smell. Furthermore, the severity of a usability smell indicates the likelyhood that the smell actually identifies a usability issue.

The following terminology is based on [3]. A user utilizes a website by performing individual, atomic *action*s. Examples for actions are a click on a button or typing text into a text field. The set of all actions executable on a website is the bounded set A. Users combine a subset of actions $B \subseteq A$ to execute a specific *task* on a website. For example, to perform the task of logging in on a website, a user combines the actions of typing his name and his password into two text fields and clicking on a confirmation button. Users can further combine tasks and actions to perform higher level tasks. For example, for performing the task of submitting a transaction on a bank website, a user performs the *subtask* of logging in on the website, the subtask for submitting the transaction, and a final action to log out by clicking on a button. We refer to the set of all tasks that can be performed on a website as the set T.

A task $t \in T$ has an ordered list of children $C_t = c_1 \dots c_n$ where each child is either an action or another task ($c_i \in (A \cup T)$). Furthermore, a task defines the execution order of its children using temporal relationships [5]. Our approach is based on the task tree generation mechanism proposed in [3], which generates the temporal relationships *sequence* and *iteration*. In case of a sequence, a task has one or more children and they are executed in their respective order. In case of an iteration, a task has exactly one child which is executed zero or more times. We refer to tasks with a temporal relationship sequence simply as sequence and to tasks with a temporal relationship iteration simply as iteration.

A task is never directly or indirectly its own child. But a single task or an action can be a child of more than one parent task. With these constraints being applied, tasks form a directed acyclic graph. For simplification, this structure is usually represented as a tree by doubling the representation of tasks occurring several times. Therefore, the directed acyclic graph is called a *task tree*. The leaf

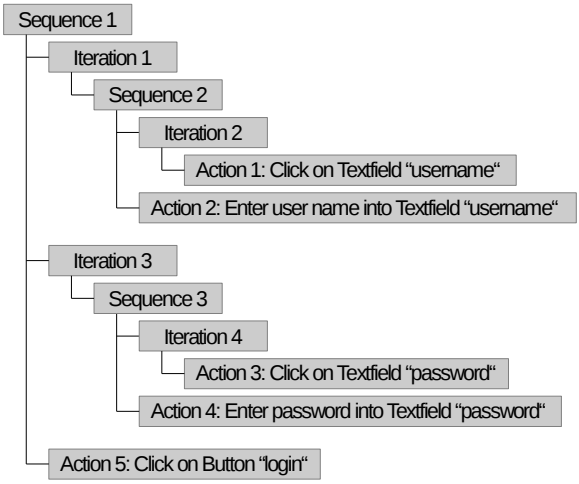

Fig. 1. Example for a task tree of a login process on a website

nodes of a task tree are the actions a user performs to complete the root task of the task tree. The intermediate nodes in the task tree divide the root task into a logical execution order of subtasks and actions.

An example for a task tree is shown in Figure 1. It represents a login process on a website. The actions are the leaf nodes. The temporal relationships of the tasks define the order in which the children have to be performed. The task (Sequence 1) starts with an iteration (Iteration 1) of a sequence (Sequence 2) for entering the user name. The user can click on the respective text field (Action 1) several times (Iteration 2). The entering of the password is modeled accordingly (Iteration 3). The overall task (Sequence 1) is completed after the user entered the user name and the password and clicks on the login button (Action 5).

The execution of an action a by a user is called an *action instance* a'. The execution of a task t is called a *task instance* t'. A task instance t' is an ordered list of action instances $a'_1 \dots a'_n$ defining the order in which the contained action instances were executed. We define $|t'|$ as the number of action instances executed in the task instance t'. A task instance must adhere to the task that was executed, i.e., the action instances and their order must match the allowed combination of actions defined by the task, its tree structure, and the temporal relationships. A task t is a representation for the set of all its instances. Therefore, we use the notation $t' \in t$ to indicate that t' is an instance of t. Furthermore, $|t|$ is the number of all instances of t.

Task and action instances contain detailed information about the task execution. For example, an action instance has a time stamp ts indicating when the action was executed. For two action instances a'_i and a'_j, we define the function $\Delta ts(a'_i, a'_j)$ that returns the time difference in milliseconds between the two action instances. Let a'_1 be the first action instance of a task instance t' and a'_n be

the last action instance of t'. Then $\Delta ts(a'_1, a'_n)$ is a measure for the execution time of t'. The smaller the execution time, the higher is the efficiency of t' and, hence, the usability of the task execution.

In [3], we describe an approach for generating task trees based on recordings of user actions on websites. The resulting task trees contain only sequences and iterations that were performed by the users and that, therefore, represent actual user behavior. We extended this approach to also dump the instances of the detected tasks in the way they were executed by the users.

3 Automated Usage-Based Usability Smell Detection

Using the generated task trees as input, we identify usability smells. We currently consider the smells *missing feedback*, *important tasks*, *required inefficient actions*, and *high website element distance*. These are described in the following subsections. For each usability smell, we provide a foundation, a description of the expected user behavior, and a description of our expectations towards the generated task trees. Furthermore, we describe the detection of the smells based on the generated task trees as well as a measure for their severity. For most smells, we also provide an example.

3.1 Usability Smell: Missing Feedback

Foundation. When utilizing a websites, users require feedback for the actions they perform [6,7,8,9]. Otherwise, they are not sure if the action has any effect and if it is processed by the system.

Expected User Behavior. Users repeat an action, if they don't see any feedback. For example, if users click on the login button during a login process and nothing visible happens, they click the same button again. Users may repeat this action even more than twice until they see some feedback.

Expected Occurrence in Generated Task Trees. Repeated actions lead to iterations in the task trees. But not all iterations of actions indicate the smell. For example, a user may repeatedly scroll a page. Therefore, we consider only a subset of iterations of actions which covers iterations of clicks on buttons and links.

Users may accidentally perform a click twice. Furthermore, they may perform a click a second time after a long period of time. For example, they may click on a download link and click the same link again to download the same file a second time. Therefore, we consider repetitions of actions as indicator of missing feedback only if they occurred within a specific time frame. This time frame may not span a full iteration of actions. For example, a user may have clicked a download link for a file three times. The first two times belong to the time frame, the third not. Therefore we consider only subsequences of repeated actions.

Let t' be an instance of an iteration of an action a and $a'_1 \dots a'_{|t'|}$ are the repeated executions of a. For each pair of action instances (a'_i, a'_j) where $1 \leq i < j \leq |t'|$, we calculate $\Delta ts(a'_i, a'_j)$. We consider an iteration of actions only, if for at least one of its instances the time difference $\Delta ts(a'_i, a'_j)$ is larger or equal to 1,000 and smaller or equal to 15,000. This means, we consider only repetitions of actions performed in a time frame of 1 to 15 seconds. All other repetitions are considered accidental multiple clicks or multiple clicks for some reason.

Detection. We filter the tasks for iterations of actions. We then consider the instances of each iteration. For each instance, we calculate a measure for the missing feedback. For this, we first subdivide the list of repeated actions $a'_1 \dots a'_{|t'|}$ into subsequences $subseq_{i,j} = a'_i \dots a'_j$ where $1 \leq i < j \leq |t'|$ and for which $1,000 \leq \Delta ts(a'_i, a'_j) \leq 15,000$. We choose these subsequences as large as possible and non-overlapping. For this, we scan the list of repeated actions for the first subsequence matching the criterion and for which $\Delta ts(a'_i, a'_j)$ is maximized but still smaller or equal to 15,000. We then search for the next subsequences starting with action instance a'_{j+1}. We repeat this searching until we find no further subsequence matching the criterion. The result of this process is a list of subsequences $subseq_{i_1,j_1} \dots subseq_{i_k,j_k}$ where k is the number of disjunctive subsequences. For the missing feedback measure of an iteration instance, we define a function $mf(t')$ as follows:

$$mf(t') = \begin{cases} \sum\limits_{subseq_{i,j}} (\Delta ts(a'_i, a'_j) \cdot (j - i)) & |subseq_{i_1,j_1} \dots subseq_{i_k,j_k}| > 0 \\ 0 & |subseq_{i_1,j_1} \dots subseq_{i_k,j_k}| = 0 \end{cases} \quad (1)$$

Through this calculation, the missing feedback measure increases with the time the user waits for repeated clicks and the number of repetitions. If the user clicks very often, the missing feedback measure increases accordingly. For each iteration of actions t, we then calculate the average missing feedback for all instances as follows:

$$mf(t) = \frac{1}{|t|} \sum_{t' \in t} mf(t') \quad (2)$$

Severity. The average missing feedback measure defines the severity of the usability smell. The usability smell is considered present if $mf(t)$ of an iteration is greater than 50ms.

Example. An example for a smelling iteration representing missing feedback is shown in Figure 2a). Iteration 1 has 3 instances. The first instance has a repetition of Action 1 (Click on Button "OK"). Based on the time stamps of the action instances, Δts for this instance was calculated to be 3000ms. For the three instances, $mf(t)$ evaluates to 1000ms. As this value is higher than 50ms, the usability smell missing feedback is present.

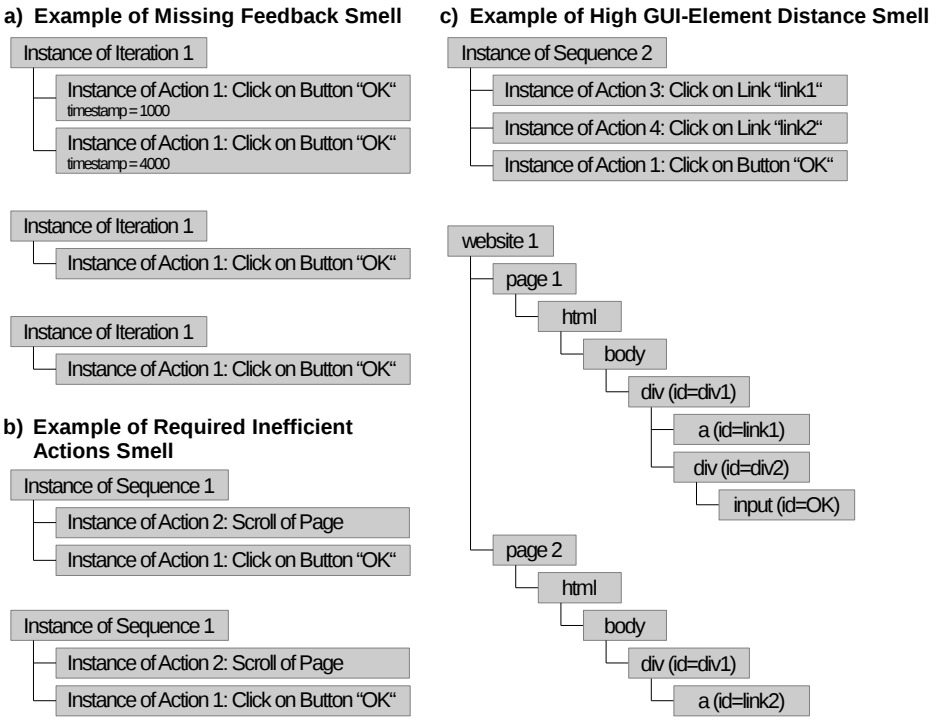

Fig. 2. Examples of smelling tasks

3.2 Usability Smell: Important Task

Foundation. Tasks performed regularly should be executed with a minimal number of actions [10,11]. If applicable, the tasks can be supported by automation, e.g., using macros.

Expected User Behavior. If users perform a task regularly, they execute always the same or similar actions.

Expected Occurrence in Generated Task Trees. Regularly performed user tasks are reflected as nodes in the generated task trees.

Detection. The task trees were generated based on the ordered set of recorded action instances RA. We search the task tree for tasks being executed most often, i.e., covering most of the recorded action instances $ra \in RA$. We only consider sequences as only sequences represent common combinations of actions and other tasks. For each sequence t, we calculate the number of all action instances covered by all instances $t' \in t$ and divide it by the number of all

recorded action instances. Through this, we get the ratio

$$r(t) = \frac{1}{|RA|} \sum_{t' \in t} |t'_i| \tag{3}$$

of action instances covered by instances of sequence t with respect to all recorded action instances. The higher this ratio, the more actions are executed in the context of task t. Therefore, the number of actions executed for task t should be minimized or automated for improving the usability.

Severity. We use the ratio $r(t)$ as indicator for the severity of the detected usability smell. The smell is considered present, if a task covers at least one percent of the recorded actions, i.e., $r(t) \geq 0.01$.

3.3 Usability Smell: Required Inefficient Actions

Foundation. Some actions are known to have no or minimal effect for fulfilling a task but also do not prevent its completion [12]. For example, when filling out a form, vertical scrolling may be required to reach the next element of the form to be filled out but does not semantically advance the process. We call such actions *inefficient actions*. The number of inefficient actions should be minimized to increase the users efficiency.

Expected User Behavior. Users perform inefficient actions during the normal usage of a website. In the case an inefficient action is required to fulfill a task, users will perform it when required.

Expected Occurrence in Generated Task Trees. As the task trees are generated based on all recorded actions, they will include inefficient actions. If an inefficient action is required at a specific point in the execution of a task, the generated task trees include this action accordingly.

Detection. For each task t, we calculate the average ratio of inefficient actions $ria(t)$. For this, we first consider each instance $t' \in t$ and determine the number of instances of inefficient actions $ia(t')$. As inefficient action, we currently consider vertical and horizontal scrolling of single panels and whole pages. Then, we calculate the average inefficient action ratio as

$$ria(t) = \frac{1}{|t|} \sum_{t' \in t} \frac{ia(t')}{|t'|} \tag{4}$$

Severity. The higher $ria(t)$, the higher is the likelyhood of task t to have a usability issue with respect to inefficiency. Therefore, $ria(t)$ is the severity of the usability smell. We consider this smell present, if $ria(t) \geq 0.1$, i.e., more than 10% of the action instances covered by instances of a task are inefficient actions.

Example. An example for the smell is shown in Figure 2b). Here, Sequence 1 has two instances. Both instances have two action instances of which one is inefficient (the scroll action). For Sequence 1, $ria(t) = \frac{1}{2}(\frac{1}{2} + \frac{1}{2})$ and evaluates to 0.5 indicating a usability smell with a high severity.

3.4 Usability Smell: High Website Element Distance

Foundation. The elements of a website required for executing a task, such as buttons and text fields, should be as colocated as possible to ensure an efficient task execution [10].

Expected User Behavior. Users utilize website elements in the order that best fits the task they use the website for.

Expected Occurrence in Generated Task Trees. The task trees show the order of website element usage.

Detection. A website consists of several pages. A page may be subdivided by several panels. Several pages may contain elements that are semantically identical. For example, all pages of a website have a menu with identical structure and menu items. We consider two distinct website elements identical if they serve the same purpose on different pages, such as a menu item. We consider two website elements close to each other, if they are on the same page in the same panel. Two website elements are less close to each other, if they are on the same page but in different panels. Furthermore, two website elements have a high distance if they belong to different pages and do not serve the same purpose. Based on this, we define a measure for the distance $d(e_1, e_2)$ of two website elements e_1 and e_2 as follows:

$$d(e_1, e_2) = \begin{cases} 0.0 & e_1 \text{ and } e_2 \text{ are identical or have the same purpose} \\ 0.2 & e_1 \text{ and } e_2 \text{ belong to the same panel} \\ 0.5 & e_1 \text{ and } e_2 \text{ belong to the same page} \\ 0.75 & e_1 \text{ and } e_2 \text{ belong to the same website} \\ 1.0 & e_1 \text{ and } e_2 \text{ belong to different websites} \end{cases} \tag{5}$$

As a panel in a page, we only consider Hyper-Text Markup Language (HTML) *div* elements as they are nowadays mostly used for structuring pages. Div elements can contain nested div elements. Furthermore, they may be freely positioned on a page using, e.g., Cascading Style Sheets (CSS). Therefore, the nesting of div elements in the HTML source code of a page is no indicator for the actual positioning on a rendered page. Hence, we consider two website elements in the same panel only if their direct parent div element is the same.

For detecting the smell, we apply the distance measure $d(e_1, e_2)$ as follows. We only consider sequences as only sequences represent common combinations

of actions and, hence, website elements. For each sequence instance $t' \in t$, we determine an ordered list of website elements in the order they were used in the sequence instance. We remove from this list the website elements used for inefficient actions (see Section 3.3) as these are not semantically required for completing a task. The result is the ordered list $E(t') = e_1 \dots e_n$ of website elements used in t'. For each subsequent pair of $E(t')$, we calculate the distances $d(e_{i-1}, e_i)$ and sum them up to the cumulative distance between all website elements used in t'. Furthermore, we determine the number of distances, which is $|E(t')| - 1$. For the whole sequence t, we sum up the cumulative distances of all sequence instances $t' \in t$ and divide it by the sum of all distances. This results in the measure $gd(t)$ which represents the average distance between two subsequently used website elements for sequence t.

$$gd(t) = \frac{\sum_{t' \in t} \sum_{i=2}^{|E(t')|} d(e_{i-1}, e_i)}{\sum_{t' \in t} (|E(t')| - 1)} \tag{6}$$

Severity. $gd(t)$ is the severity of the usability smell. We consider this smell present if $gd(t)$ for a sequence is higher than 0.5, i.e., if in average two subsequently used website elements are not on the same page.

Example. An example for the smell is shown in Figure 2c). The tree on the top is an instance of a sequence called "Sequence 2". The tree on the bottom represents the structure of the corresponding website (root node). The website has 2 pages (children of the root node) which consist of several HTML tags and their respective structuring (all other nodes). An HTML tag may have an id to differentiate two HTML tags having the same name.

The instance of Sequence 2 consists of three action instances. The action instances utilize $|E(t')| = 3$ elements of two different pages of the website: first "link1" on the first page, then "link2" on the second page, and finally button "OK" on the first page again. The cumulative distance of the website elements for this instance is 1.5 and the number of distances is 2. As this is the single instance of Sequence 2, the resulting distance measure for Sequence 2 is $gd(t) = \frac{1.5}{2}$. This value is greater than 0.5 and, therefore, a usability smell is present.

4 Case Studies

We implemented our approach based on the tool suite for Automatic Quality Engineering of Event-driven Software (AutoQUEST) [13]. In addition to others, AutoQUEST allows recording user actions for different platforms. We utilized AutoQUEST's ability to record user actions on websites. For this, each page of a website must include a JavaScript provided by AutoQUEST. The recorded action instances are sent to a dedicated AutoQUEST server which stores them intermediately. Afterwards, they are fed into AutoQUEST for further processing.

AutoQUEST provides the mechanism for generating task trees based on recorded user actions which we described in [3]. As explained in Section 2, we extended this mechanism with the ability to generate task instances. Furthermore, we implemented the detection of the usability smells described in the previous section. Finally, we used this implementation in two case studies.

4.1 Description of the Case Studies

In the first case study, we applied our approach on the website of our research group [14] at the University of Göttingen. This website offers information about the members of our research group (e.g., contact information), details and materials of lectures we offer to students, as well as details of our research including publication lists. Members of the research group can login on the website to edit the content. The website offers 18 different types of pages. As type of page, we consider several pages having the same structure. For example, all details pages with information about the group members have the same structure and, therefore, are of the same page type.

The website is based on a content management system in which we integrated AutoQUEST's mechanism for recording user actions. As a result, we recorded 33,690 action instances over a period of 8.5 months. The task tree generation generated 1,847 tasks of which 1,431 were sequences and 416 were iterations. Further details of the case study can be found in Table 1.

Table 1. Facts of the case studies

	Case Study 1 Website of Research Group	**Case Study 2** Application Portal
Page types	18	35 (21 application part, 8 review part, 6 shared)
Recording period	8.5 months	4.5 months
Start	July 2013	October 2013
End	March 2014	March 2014
Recorded actions	33,690	241,306
Generated tasks	1,847	10,634
Sequences	1,431	9,530
Iterations	416	1,104

In the second case study, we applied our approach on a master studies application portal on our university. This portal is subdivided into two major parts. The first part allows student applicants to apply for master studies at our university. In a wizard based fashion, applicants are asked to enter their personal data and information about previous studies and achievements. At the end of

the wizard, they get an overview of the entered data in which they are asked to upload their CV, a photo and certificates, e.g., about former studies. In the second part of the portal, a reviewer team can verify the applications and accept or reject them including entering reasons. The portal offers a login mechanism including functions for registration and changing the password.

The application portal is based on a content management system in which we integrated AutoQUEST's mechanism for recording user actions. As a result, we recorded 241,306 action instances over a period of 4.5 months. The task tree generation generated 10,634 tasks of which 9,530 were sequences and 1,104 were iterations. Further details of the case study can be found in Table 1.

The number of reported usability smells in both case studies and their distribution to the different smell types is shown in Table 2. Our approach reported 5,900 smells of which 1,464 were found in the first, and 4,436 were found in the second case study. These were too many to be interpreted manually. Furthermore, many findings referred to tasks that cover only a small set of recorded action instances or that were executed only seldom. Hence, we considered these smells as less valuable. Therefore, we performed a preselection of the detected smells based on a measure for the importance of the tasks for which the smells were found. Similar to the respective smell, we considered a task more important, the more recorded action instances it covers. We selected a variable percentage q of tasks that cover the most recorded action instances and being, therefore, most important. Then we selected only those smells that referred to the selected tasks. We adjusted q so that for both case studies at most 100 smells resulted from the selection. Setting q to 5%, i.e., selecting only smells for those 5% of tasks that cover most of the recorded action instances, resulted in 96 detected smells for the first case study and 75 smells for the second case study.

Table 2. Smells detected in the case studies

	Case Study 1 Website of Research Group			**Case Study 2** Application Portal		
	all	redundant	true positive	all	redundant	true positive
Detected smells	1,464	275		4,436	985	
Missing feedback	52	0		39	0	
Important task	19	10		13	4	
Inefficient actions	959	199		3,772	878	
High distance	434	66		612	103	
Considered smells	96	30	83	75	22	64
Missing feedback	12	0	11	9	0	6
Important task	19	10	10	13	4	11
Inefficient actions	54	17	51	51	18	47
High distance	11	3	11	2	0	0

In both case studies, several smells of the same type were reported for a task and also for one or more of its children. We call these reports *redundant findings*. We considered them as a hint to better analyze the cause of a finding for a parent task. The number of redundant findings for each smell is shown in a dedicated column in Table 2.

4.2 Findings for the Smell Missing Feedback

The detection of missing feedback (Section 3.1) revealed 91 tasks with usability smells of which 52 were found in the first case study and 39 in the second. There were no redundant findings. Based on the filtering we considered 12 smells for the first case study and 9 for the second. The smells referred to the following repetitions of clicks:

- First case study
  - clicks on 9 links for navigating inside the website (e.g. the menu)
  - clicks on 2 links for navigating inside the publication list
  - clicks on 1 download link for a file
- Second case study
  - clicks on 6 links and buttons concerned with user registration and login
  - clicks on 2 buttons used for navigating back and forth in the wizard
  - clicks on 1 button used to download and view an uploaded file

4.3 Findings for the Smell Important Task

The detection of important tasks (Section 3.2) resulted in 32 tasks with 14 redundant findings. 19 smells with 10 redundant findings were found in the first case study and 13 smells with 4 redundant findings in the second. The filtering of the smells did not filter any of the detected smells. This is due to the fact, that the filter for the smells and the filter for important tasks use the same mechanism and that the filter for the smells is less restrictive. Therefore, we considered all found smells for this smell type. The smells referred to the following types of task, that overlap each other:

- First case study
  - 9 tasks that included at least one inefficient scrolling action
  - 6 tasks including the navigation via the menu of the website
  - 6 tasks concerned with the login process
  - 5 tasks for showing the overview and details of the lectures offered by our group
  - 4 tasks for showing details of our group members
  - 2 tasks for filtering the list of publications of our group
  - 2 tasks for showing details of our research in general
- Second case study
  - 2 tasks that included at least one inefficient scrolling action
  - 6 tasks representing the utilization of a date chooser in the wizard
  - 3 tasks concerned with the login process
  - 2 tasks for entering details about former studies
  - 1 task for navigating in the wizard
  - 1 task for uploading files

4.4 Findings for the Smell Required Inefficient Actions

The detection of required inefficient actions (Section 3.3) resulted in 4,731 findings of which 1,077 were redundant. 959 smells with 199 redundant findings were found in the first case study and 3,772 smells with 878 redundant findings in the second. The filtering of the smells reduced the amount of considered smells to 54 with 17 redundant findings in the first case study and 51 with 18 redundant findings in the second case study. The smells referred to the following types of tasks:

- First case study
  - 26 tasks including the usage of the menu
  - 19 tasks for showing the overview and details of the lectures offered by our group
  - 11 tasks for showing details of our group members
  - 9 tasks concerned with the login process
  - 7 tasks for filtering the list of publications of our group
  - 4 tasks for showing details of our research in general
  - 2 tasks for editing content
  - 1 tasks for showing news of our research group
- Second case study
  - 15 tasks for providing application details during the usage of the wizard
  - 12 tasks for uploading files
  - 7 tasks for showing details of a specific application on the reviewer part
  - 6 tasks concerned with the login process
  - 5 tasks for navigating in the wizard
  - 5 tasks representing the utilization of a date chooser in the wizard
  - 4 tasks for providing review details and rejection reasons on the reviewer part
  - 2 tasks for submitting and revoking an application

4.5 Findings for the Smell High Website Element Distance

The detection of high website element distances (Section 3.4) found 1,064 usability smells of which 169 were redundant. 435 smells with 66 redundant findings were found in the first case study and 612 smells with 103 redundant findings in the second. The filtering of the smells reduced the amount of considered smells to 11 with 3 redundant findings in the first case study and 2 with no redundant finding in the second case study. The smells referred to the following types of tasks:

- First case study
  - 10 tasks for showing the overview and details of the lectures offered by our group
  - 9 tasks including the usage of the menu
  - 3 tasks concerned with the login process
- Second case study
  - 1 task for navigating in the wizard
  - 1 task for showing details of a specific application on the reviewer part

5 Discussion

Our case studies show, that our approach finds usability smells. The number of detected smells that indicate true usability issues are listed in Table 2 in the "true positives" columns. To decide, which of the smells are true positives, we performed a manual check for each of the filtered smells. However, the severity of several found smells is rather low. This should be addressed by adjusting the lower boundary of the severity for each smell which must be exceeded to consider a finding. Furthermore, a ranking of the smells is required based on their severity.

The detection of missing feedback shows good results. We consider 17 of the 21 smells detected in both case studies as real usability issues. The findings for internal navigation on our research website are inline with our observations. During the recording period, our research website had a performance problem which may have caused users to click several times on the same link. We addressed this issue by adjusting the configuration of the content management system. We also consider the findings for the login process in the second case study as usability issues, as we also here experienced performance issues. The findings for the next and previous buttons in the wizard of the application portal are false positives as the users utilized these buttons subsequently to navigate between several wizard views. The findings of missing feedback for file downloads in both case studies could not be confirmed. The downloads were usually fast enough and the progress was indicated by the web browser. These findings may show, that the 1 and 15 seconds boundaries used in the smell detection require an adjustment.

The findings for important tasks are helpful in both case studies. We consider 21 of 32 as true positives. The other findings referred to tasks that included only one efficient action and, therefore, did not represent useful action combinations. Based on the findings for our research website, we saw findings for five tasks representing navigations from the start page via the list of lectures to details of a specific lecture. This shows that details of a specific lecture are often required. We now consider to put links to ongoing lectures to the start page to reduce the number of clicks required for accessing this information. The same applies to details of our group members and the login process. For the application portal, the usage of the date chooser is most important. Therefore, we consider to optimize it further to reduce the number of required actions.

The findings for required inefficient actions showed many redundant findings (35 of 105). However, the findings were helpful and we consider 98 of them as usability issues. The others are caused by the fact, that users vertically scrolled pages with a lot of information which are intended to be read or to display an overview. And vertical scrolling of pages while reading is not a usability issue. For our research website, we consider to restructure the pages so that the size of the top banner decreases on all pages but the start page. This would allow to move content of details pages to move upwards on the page. Furthermore, we consider to move the menu from the top of the page to the left and prevent its scrolling out of sight. Through this, we hope to reduce the number of required scrolling for navigating on the website and its details pages. For the application

portal, especially the scrolling during the wizard usage was important. A wizard should break down a task into simple subtasks. The required scrolling indicates that still the subtasks are quite complex, require many website elements and, hence, much screen space. Therefore, we consider to check the wizard and the presented forms for their complexity. If required, we will further break down some subtask. Furthermore, scrolling after a file upload was important. This issue is caused by the fact that for a file upload, the current page reloads and starts at the top again. The users have to scroll the page to get to the next file to be uploaded. This issue was already tried to be addressed by adding a link on the top of the page allowing the automatic scrolling to the next file to be uploaded. But still, this is inefficient. Therefore, we propose to change the file uploads to make them possible without requiring scrolling.

The findings for high website element distances are mostly inline with the findings of other smells. We consider 11 of the 13 smells detected in both case studies as real usability issues. In the research website case study the findings showed that users have to navigate via several pages to reach details of a lecture or group member. Therefore, the website element distance is relatively high. Hence, these findings support, e.g., the findings for important tasks. The findings for the second case studies are not considered usability issues. Both findings represent required navigation between pages either for drill down in a large set of information (finding for the reviewer part) or for navigating in the wizard.

Our approach relies on a large number of recorded user actions. Otherwise, the task trees would not represent real user behavior. Therefore, the approach can only be applied on large data sets recorded over a long period of time. Based on the observations in the case studies, the validity of our results seems relatively high. We consider 147 of 171 detected usability smells as indicators for usability issues and, hence, as helpful findings. However, the validity should further be confirmed using other, established usability evaluation methods, e.g., usability tests with thinking aloud, in a comparative study. Furthermore, we should apply our approach in further case studies to determine if we obtain similar results. For example, we need to clarify if the minimum severity used for each smell is also adequate for smaller websites or websites with different structures.

6 Related Work

The term usability smells is similar to code smells introduced by Fowler [15]. Code smells are structures of source code being a hint for a required refactoring. Similarly, usability smells are usage patterns indicating required usability improvements.

Ivory and Hearst [16] introduced a categorization for usability evaluation methods with respect to automation. The categories are *capture*, *analysis*, and *critique*. Depending on the level of automation, an evaluation method is assigned to one or more of these categories. According to Ivory and Hearst, there are no methods yet covering all these categories at the same time. Our method spans all categories in that it records user actions (capture), transforms the recorded data

into task trees representing actual user behavior (analysis), and automatically reasons on aspects of the websites usability (critique).

There are several methods that record users when utilizing a website and subsequently analyze the data to detect usability problems. Examples are WebHint [17], WebRemUsine [18,19], and ReModEl [20]. With respect to Ivory and Hearst [16], these methods belong to the groups capture and analysis. The methods compare recorded user actions with expected user behavior. The expected user behavior is either a recording of actions of an example session (WebHint) or a manually defined task model (WebRemUsine, ReModEl). The methods only show differences between expected and recorded user behavior. Therefore, they do not belong to Ivory and Hearsts critique category. In comparison to these methods, our approach does not consider expected user behavior. Instead, our approach assesses only real user behavior. This has the additional benefit that no wrong expectations on user behavior influence our analysis. Furthermore, our approach provides details about the location and causes of potential usability issues and does, therefore, belong to Ivory and Hearsts critique category.

John et al. [21] propose an approach to record user actions and generate Goals, Operators, Methods, and Selection Rules (GOMS) models based on them. The GOMS models are used to estimate the average efficiency of users for executing a specific task. In comparison to our approach, this method focuses on efficiency measurement and is only used for example sessions performed by an evaluator. Sessions of real users are not considered.

An approach that belongs to Ivory and Hearsts critique category is the Mental Model workbench (MeMo) [22] and its combination with the Multi-Access Service Platform (MASP) [23]. In this concept, a Graphical User Interface (GUI) is rendered and an analyzer tool, which simulates a user, evaluates which action is most likely to be chosen by the user in a specific state of the GUI. An extended and iterative version of this approach is proposed by Quade [24] where the evaluation also performs GUI adaptations for usability improvement. The goal is to iteratively design a GUI guiding users as good as possible in the execution of specific tasks. In contrast to our approach, this technique works at design time to improve a GUI under development. It does not consider real user behavior but simulates it. Furthermore, it requires predefined user tasks and does not expect users to apply a GUI for unexpected tasks.

7 Conclusion

In this paper, we presented an automated approach for detecting usability smells of websites based on recorded user actions. The detected types of usability smells are missing feedback, important tasks, required inefficient actions, and high website element distance. The approach was applied in two case studies. For all smell types, we gained helpful results but also some room for improvement. For example, the detected smells included redundant findings and false positives. As the majority of the detected smells that we considered in our analysis also indicated usability issues, we conclude that it is possible to detect usability issues using our approach.

In our future work, we plan to improve the described usability smell detection, e.g. to reduce the number of redundant findings. In this context, we will also perform detailed studies with different detection parameters such as the minimal severity that must be reached by a smell to be reported. Furthermore, we will perform studies for specific smell detections with adjusted algorithms to check whether a modified algorithm provides better results. For example, we plan to implement the high website element distances detection without excluding inefficient actions and compare the results with our current findings. Finally, we plan to implement more usability smell detections, e.g. to detect unused but prominently positioned website elements which may also require an improved task tree generation to allow for more sophisticated findings.

References

1. Sarodnick, F., Brau, H.: Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung, 1st edn. Huber, Bern (2006)
2. Krug, S.: Web Usability: Rocket Surgery Made Easy. Addison-Wesley (2010)
3. Harms, P., Herbold, S., Grabowski, J.: Trace-based task tree generation. In: Proceedings of the Seventh International Conference on Advances in Computer-Human Interactions (ACHI 2014). XPS - Xpert Publishing Services (2014)
4. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability (ISO 9241-11:1998), ISO (1998)
5. Paternò, F.: ConcurTaskTrees: An engineered approach to model-based design of interactive systems. In: The Handbook of Analysis for HumanComputer Interaction, pp. 1–18 (1999)
6. Polson, P.G., Lewis, C., Rieman, J., Wharton, C.: Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. Int. J. Man-Mach. Stud. 36(5), 741–773 (1992)
7. Ferré, X., Juristo, N., Windl, H., Constantine, L.: Usability basics for software developers. IEEE Softw. 18(1), 22–29 (2001)
8. Norman, D.A.: The design of everyday things, 1st edn. Basic Books, New York (2002)
9. Balbo, S., Goschnick, S., Tong, D.: Leading Web Usability Evaluations to WAUTER. In: The Eleventh Australasian World Wide Web Conference. Gold Coast (2005)
10. Lecerof, A., Paternò, F.: Automatic support for usability evaluation. IEEE Trans. Softw. Eng. 24, 863–888 (1998)
11. Tidwell, J.: Designing Interfaces - Patterns for Effective Interaction Design. In: Treseler, M. (ed.), 2nd edn. Oreilly Series. O'Reilly Media, Incorporated (2010), http://books.google.de/books?id=5gv0U9X0fu0C
12. Patern, F., Piruzza, A., Santoro, C.: Remote usability analysis of multimodal information regarding user behaviour, pp. 15–22 (2005), http://giove.isti.cnr.it/attachments/publications/2005-A2-134.pdf
13. Herbold, S., Harms, P.: AutoQUEST - Automated Quality Engineering of Event-driven Software (March 2013)
14. Software Engineering for Distributed Systems Group. Software Engineering for Distributed Systems (2014), http://www.swe.informatik.uni-goettingen.de/ (retrieved: 4, 2014)

15. Fowler, M.: Refactoring: Improving the Design of Existing Code. Addison-Wesley, Boston (1999)
16. Ivory, M.Y., Hearst, M.A.: The state of the art in automating usability evaluation of user interfaces. ACM Comput. Surv. 33, 470–516 (2001), http://doi.acm.org/10.1145/503112.503114
17. Vargas, A., Weffers, H., da Rocha, H.V.: A method for remote and semi-automatic usability evaluation of web-based applications through users behavior analysis. In: Proceedings of the 7th International Conference on Methods and Techniques in Behavioral Research, MB 2010, pp. 19:1–19:5. ACM, New York (2010), http://doi.acm.org/10.1145/1931344.1931363
18. Paganelli, L., Paternò, F.: Tools for remote usability evaluation of web applications through browser logs and task models. Behavior Research Methods 35, 369–378 (2003)
19. Paternò, F., Russino, A., Santoro, C.: Remote evaluation of mobile applications. In: Winckler, M., Johnson, H., Palanque, P. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 155–169. Springer, Heidelberg (2007)
20. Buchholz, G., Engel, J., Märtin, C., Propp, S.: Model-based usability evaluation – evaluation of tool support. In: Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 1043–1052. Springer, Heidelberg (2007)
21. John, B.E., Prevas, K., Salvucci, D.D., Koedinger, K.: Predictive human performance modeling made easy. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2004, pp. 455–462. ACM, New York (2004)
22. Jameson, A., Mahr, A., Kruppa, M., Rieger, A., Schleicher, R.: Looking for unexpected consequences of interface design decisions: The memo workbench. In: Winckler, M., Johnson, H. (eds.) TAMODIA 2007. LNCS, vol. 4849, pp. 279–286. Springer, Heidelberg (2007)
23. Feuerstack, S., Blumendorf, M., Kern, M., Kruppa, M., Quade, M., Runge, M., Albayrak, Ş.: Automated usability evaluation during model-based interactive system development. In: Forbrig, P., Paternò, F. (eds.) HCSE/TAMODIA 2008. LNCS, vol. 5247, pp. 134–141. Springer, Heidelberg (2008)
24. Quade, M., Blumendorf, M., Albayrak, S.: Towards model-based runtime evaluation and adaptation of user interfaces. In: User Modeling and Adaptation for Daily Routines: Providing Assistance to People with Special and Specific Needs (2010)