Check for updates

Automated Usability Evaluation of Virtual Reality Applications

PATRICK HARMS, University of Goettingen

Virtual reality (VR) and VR applications have reached the end-user and, hence, the demands on usability, also for novel applications, have increased. This situation requires VR usability evaluation methods that can be applied quickly, even after a first release of an application. In this article, we describe such an approach, which is fully automated and does not ask users to perform predefined tasks in a fixed test setting. Instead, it works on recordings of the actual usage of a VR application from which it generates task trees. Afterwards, it analyzes these task trees to search for usability smells, i.e., user behavior indicating usability issues. Our approach provides detailed descriptions of the usability issues that have been found and how they can be solved. We performed a large case study to evaluate our approach and show that it is capable of correctly identifying usability issues. Although our approach is applicable for different VR interaction modalities, such as gaze, controller, or hand interaction, it also has limitations. For example, it can detect diverse issues related to user efficiency, but specific misunderstandings of users cannot be uncovered.

CCS Concepts: • **Human-centered computing** → **Usability testing**; *User models*; *User studies*; *Laboratory experiments*;

Additional Key Words and Phrases: Automated usability evaluation, task tree generation, usability smell detection, virtual reality

ACM Reference format:
Patrick Harms. 2019. Automated Usability Evaluation of Virtual Reality Applications. *ACM Trans. Comput.-Hum. Interact.* 26, 3, Article 14 (April 2019), 36 pages.
https://doi.org/10.1145/3301423

1 INTRODUCTION

Virtual Reality (VR) will gain more relevance in the upcoming years [22]. If it has a similar development as mobile phones and apps, then many new applications will flood the market. Companies will have an interest of reaching early adopters and doing a first release of their applications as fast as possible to be technological pioneers. For the success of VR in the end-user market, the apps need to provide a high usability, which can be achieved through usability engineering. But an envisaged short time-to-market may contradict with high efforts for usability engineering. In addition, not all VR developers have a usability engineering background. And even if they have, usability evaluations of VR may require new techniques dedicated to VR as consumer product [14].

To address this situation, we propose a simple and fast usability evaluation technique for VR. The technique is fully automated and performs the following three steps:

Author's address: P. Harms, Institute of Computer Science, University of Goettingen, Goldschmidtstraße 7, Goettingen, Lower Saxony 37077, Germany; email: patrick.harms@cs.uni-goettingen.de.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
1073-0516/2019/04-ART14 $15.00
https://doi.org/10.1145/3301423

—Record the usage of a VR by logging user actions.
—Generate task models in form of task trees from the recordings.
—Detect usage pattern in the task trees that indicate usability issues.

Through the automation, we want to provide VR developers with a method for technicians that does not require too much usability engineering background and can be applied even if apps are released and updated in short development cycles. As we record the usage of a running VR, the approach is applicable for later development stages and between app releases. In general, the approach can be the sole method of usability engineering performed for a VR app, but due to its limitations we uncover in this article, we propose it as an extension to other usability engineering efforts taken for VR development.

In contrast to traditional user-oriented usability evaluation, our approach is not based on benchmark tasks that the users are intended to perform within a VR. This is to make the approach applicable without any further preparation, e.g., without deriving user tasks using a contextual inquiry. Instead, the users use the VR for self-selected tasks in the way that best fits their needs. Through the generation of the task trees, our approach determines the user tasks on its own.

The usage patterns we detect in the third step of the approach are called *usability smells*. The corresponding findings include details on the location of the underlying usability issues as well as proposals for their solving. Hence, the approach goes a step further than simple VR analytics, where these conclusions need to be drawn by an expert. Therefore, the analysis of the results of our approach should be rather straightforward making it applicable for everybody, not only usability experts.

In this article, we focus on the validity of the results of our proposed approach and determine its limitations. We leave the analysis of the applicability of the approach by non-usability experts to future research. More concretely, we answer three research questions. Each of them focuses on the individual steps of our proposed approach. They are as follows:

—RQ 1: How and to what extent must the usage of a VR be recorded and stored for a subsequent task tree generation?
—RQ 2: How and to what extent can task trees be generated for recordings of VR usage?
—RQ 3: How and to what extent can usability issues in a VR reliably be derived from generated task trees?

This article is structured as follows. First, we introduce important terminology in Section 2 followed by related work in Section 3. Then, we describe our approach with all required information for a reimplementation in Section 4. Afterwards, we provide details of the case study that we performed to validate our approach and discuss its results and threats to validity in Section 5. In the final Section 6, we subsume the article and propose future work.

2 FOUNDATIONS

In this section, we introduce diverse terms which are of relevance for this article. This covers the topics VR, user actions, tasks, task trees, events, and user action recording.

2.1 Virtual Reality

VR is a technique for letting users observe and interact with a *virtual world* [24]. A virtual world can be seen as an observable and changeable graphical user interface (GUI) of a computer system [24]. As such, it puts the interaction with computer systems to a different level than traditional GUIs for websites or desktop programs do. For example, in addition to, or as replacement of, buttons and text fields, the interactive elements can be three-dimensional (3-D) objects like knobs and wheels.

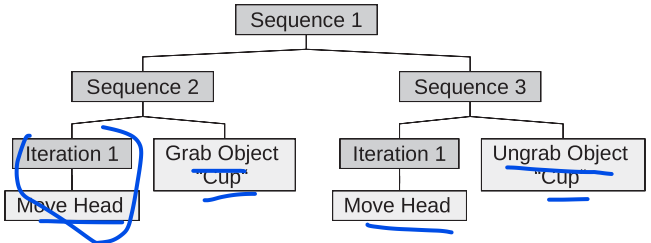

Fig. 1. Example of a simplified task tree.

The enabling factor of a VR is a corresponding system, a *VR system*. With the technological development in the last decades, also the capabilities of VR systems for the consumer market increased making the experience of virtual worlds more realistic. This includes stereoscopic head-mounted display and room-scale tracking. For interacting with a virtual world, the most prominent consumer market VR systems, e.g., the HTC Vive [19], the Oculus Rift [64], or Microsoft's Windows Mixed Reality Headsets [61], require users to utilize controllers. There are also attempts to track user movements, e.g., using a Leap Motion [55] for tracking the user hands. Through this, the users can see and use virtual representations of their hands in the virtual worlds. But due to missing haptic feedback, this still does not feel fully natural for users.

2.2 Actions, Tasks, and Task Trees

If not stated otherwise, the following descriptions are based on our previous work [37–41]. When users interact with a software, they perform individual *actions,* such as clicking on a button or grabbing an object in a VR. Most of these actions directly contribute to the semantic fulfillment of a task. We, therefore, call them *effective actions*. There are also actions which may not contribute to the task, like scrolling a web page or moving the head in a VR. We call them *inefficient actions* as they only require time.

When users use a software, their intention is to fulfill a *task*, e.g., performing a transaction on a bank website or manipulating a 3-D prototype of a car in a VR. A task can be subdivided into *subtasks*, e.g., the subtask of logging in into a system or entering a virtual room before other subtasks can be done. Subtasks can also be children of other subtasks, i.e., tasks and subtasks form a hierarchical structure.

To perform a subtask, users execute multiple actions. These can be manifold and their ordering is not necessarily fixed for a certain subtask. To be able to model a task, so called *task models* can be used. They describe in which order actions must be executed to fulfill subtasks and tasks [92]. A certain variant of task models are *task trees*, e.g., the ConcurTaskTrees proposed by Paternò [68, 70]. A simplified variant of task trees is shown in Figure 1. This task tree describes the grabbing and releasing of an object in a VR. The leaf nodes are the actions that users need to do for this task. The parent nodes define the order in which the actions are taken. There are two types of parent nodes, *iteration,* and *sequence*. An iteration has only one child which can be repeated multiple times. For example, in Figure 1, *Iteration 1* denotes that users can move their heads multiple times. A sequence represents a subtask and has multiple children which are executed in the given order. For example, in Figure 1, *Sequence 2* defines that the users first move their head before grabbing the object "Cup." In addition, *Sequence 1* defines that users first grab the object before they release (ungrab) it elsewhere.

2.3 Events and Action Recording

Actions cause *events* in a software which the software processes to react on the actions [37]. For example, for reacting on actions of users of a website, the website internally processes events, such

as clicks or text inputs. The same applies for VRs. The main difference to traditional 2-D GUIs is that the types of events are different. In contrast to events resulting from clicking or scrolling, the users cause events for grabbing objects or moving their heads.

To record the usage of a software, it is common to extend the internal event handling mechanisms with a journaling of the events into a log file or database (e.g., [5] or [21]). Conceptually, the result of such a recording are *event lists*. These contain the recorded events in the order in which they occurred. As the individual events represent executed user actions, event lists represent the usage of software. When recording the events caused by multiple users of a software, the results are multiple event lists, one for each usage of the software.

3 RELATED WORK

In this section, we describe other research related to our work. We first put our work into the context of usability engineering. Then, we consider other attempts of automating usability evaluation. Finally, we focus on each step of our approach and reference related work in similar directions. Our own approach reuses and adapts existing techniques, e.g., for task tree generation and usability smell detection. These techniques are referenced and described in Section 4 to ensure a complete and consistent description of our approach in a single section. Hence, we do not reference this work in the following subsections.

3.1 Usability Engineering and Automation

Our work focuses on the assessment of VR *usability*. We use the definition of this term from ISO 9241-11 [1]. There, usability is the matching of a software product with its user groups, their tasks, and the context of the tasks' execution. If this is given, then users perform their tasks with *effectiveness*, *efficiency*, and *satisfaction*. Usability is a subset of *user experience* [90] which additionally takes into account the broader situation of the user. To ensure a high usability of a software, there are diverse methods for measuring different usability aspects. These methods and their application are subsumed under the term *usability engineering* [60, 78, 83]. The approach described in this article intends to be a further method under the umbrella of this term.

The methods in usability engineering can be subdivided into user- and expert-oriented methods. In user-oriented methods, potential users of a software perform predefined tasks with the software [62] while being observed by usability experts. In expert-oriented methods, usability experts perform a set of predefined steps with the goal of a formal assessment of a certain usability aspect [62]. Both types of usability engineering methods require a high manual effort. In addition, while there is much experience with these methods for 2-D interfaces, there has been only limited work evaluating their applicability for VRs [14, 79]. The method presented in this article is similar to user-oriented methods as it is based on users using the VR. But in contrast to existing methods, the users do not perform predefined tasks, but use the software as best fits their needs. As users also do not need to be aware of being observed, they do not feel in a test situation what may influence the test results [43]. In addition, our method does not require an evaluator to observe the user. The observation is fully automated by recording the user actions. Furthermore, the observations do not need to be analyzed manually by an evaluator, but the analysis is automated as well.

In the recent years, there have been several attempts for automating usability engineering. A well-known and established framework to characterize these attempts is the literature review by Ivory and Hearst [47]. They identified three usability engineering automation levels being capture, analysis, and critique. Capture means automation with respect to recording the usage of a software. Analysis aims at identifying usability issues based on the recorded data. Critique provides proposals for solving these issues. The approach that we describe in this article provides capture, analysis, and critique in one process considering the special requirements of VR applications.

A further term that evolved in the last decades is *remote usability evaluation* for web and mobile applications [69, 71–73]. This is similar to user-oriented usability evaluation as users are asked to perform certain tasks with a software. The main difference is that the interaction is done separated from the evaluator in space, and probably also in time [7]. For the usage analysis, the user actions are recorded and later compared with expected interactions. The expected interactions can be defined through a manually created model [42] or an example execution [15, 16]. Also in this approach, many parts can be automated. But still the approach requires some manual preparation, like task analysis and definition of expected interactions. Furthermore, the evaluator has to derive why users behave differently than expected and what is the corresponding usability issue. Our approach neither requires an extensive manual preparation nor a manual analysis of results.

The idea of utilizing recordings of user action events for a subsequent usability analysis is no new topic and already considered for other types of software than VR. Already in 2000, Hilbert and Redmiles defined a framework with the aim to classify different approaches that perform usability evaluations based on recorded user actions [44]. Their framework differentiates available methods into the groups *synchronization and searching*, *transformation*, *analysis,* and *visualization*. They further consider *integrated support*, which are methods that belong to all of the before mentioned groups. The first group, synchronization and searching, considers methods that allow to match recorded user action streams to video recordings or other recorded data with the goal of combining multiple information sources as a basis for usability analysis. Our approach does not belong to this group as it is based on recorded user actions, only. The second group, called transformation, considers methods that perform post-processing of the recordings, e.g., by dropping unnecessary data or abstracting from low-level recordings to higher level user tasks, which may be required to clean up the recorded data. Our approach belongs to this group as the recordings of user actions in VR are on a rather low key stroke level. Hence, it requires to filter out irrelevant actions and to combine relevant action combinations, e.g., a hand movement in combination with a gaze, to higher level actions and task, e.g., to a certain interaction with an object in a VR. The next method group of Hilbert and Redmiles is analysis. This is further subdivided into the provisioning of plain usage statistics and the detection of typical action sequences. While in our approach plain usage statistics will be a by-product, the detection of typical action sequences and their characterization will be most fundamental for the whole approach. The detection of typical action combinations, their comparison, and their characterization to derive task models is one of the most important aspects of our approach. The last group of methods identified by Hilbert and Redmiles is visualization, which focuses on presenting recorded data to an evaluator. Although not being the major goal of our approach, it provides possibilities to visualize the recorded data as well as the task trees generated from it. In addition, our approach provides detailed representations of the usability smells and issues that are found with it as well as proposals for solving them.

An orthogonal direction to automation of usability evaluation is static analysis of GUIs. Here, the internal structure or the rendered variant of a GUI is analyzed and checked for anomalies with respect to established guidelines. This comprises plain metric calculation and checking for conformance to guidelines (e.g., HUIA [6, 8], USEFul [23], and Google's Mobile-Friendly Test [32]), and model-based simulation of GUI usage (e.g., MeMo Workbench [48]). Some approaches also attempted to automate user-oriented evaluation techniques directly on the GUI model (e.g., InfoScent [52]). The approach proposed in this article does not perform a static GUI analysis, but focuses on analyzing user behavior.

3.2 Recording Software Usage

The first step of our approach is recording user actions in a VR. Basically, recordings can be done in different ways and on different levels starting from video recording, via screen casting up to

logging software events [80]. In our approach, we perform a logging of events caused by the utilization of a VR.

Two-dimensional user interfaces, such as GUIs are created using dedicated frameworks. These frameworks come with an established set of interaction elements like buttons, text fields, and so on, as well as corresponding event handling mechanisms. This applies to desktop frameworks (e.g., Java Swing [67]) and mobile frameworks (e.g., Android [33]) as well as to websites (e.g., HTML 5 [96], Bootstrap [13]). There are already diverse implementations of logging mechanism for these and other frameworks, e.g., for desktop applications based on Quicktime [17], Windows [47, 77], Java [76], and games [53, 97]. Other related work shows ways of recording apps on mobile devices [49] and the usage of websites [5, 21, 36]. For VR applications, there are also frameworks for object creation and event handling. Examples are the GUI framework provided with the game engine Unity 3-D [91] and the VR Toolkit [2]. To the best of our knowledge, there is no established user interaction recording mechanism for VR yet, except the one we describe in this article.

When considering recording actions of users, very often the term *analytics* with tools like Matomo [58] and Google Analytics [31] for websites comes into play. Similar techniques exist for mobile devices and games [97]. The goal there is to trace users and their actions on a website or other software and calculate a broad variety of metrics. These metrics can then help to understand user behavior if interpreted correctly. But the level of detail on which these recordings take place or the accessibility of data do not allow for a more in-depth analysis. In our work, we go into more details for an analysis of the recorded usage of a VR, for example, by reconstructing the user tasks. We record the usage in more detail than typical analytics systems do and we provide more than plain statistics on the usage of a VR.

3.3 Task Tree Generation

The next step of our approach is to generate task models in form of task trees from recorded VR user actions. Van Welie et al. [92] defined an ontology for task models to define a coherent terminology to be used with task modeling. Following this ontology, task models describe four things:

(1) The structure of a task through a decomposition into subtasks and actions.
(2) The flow of a tasks, i.e., the order in which the actions of a task are executed.
(3) The objects that are required for the task execution.
(4) The environment of the task execution, the so called task world.

The task models that we generate in our approach cover the first two items. To a certain extent, also the third item is addressed as the recorded actions have a reference to the utilized VR and the elements with which an interaction took place. But Van Welie et al. also consider additional real-world objects, which are not referenced in our recorded data and which can, therefore, also not be represented in task models resulting from this data. The same issue applies for the fourth item.

Task trees are an established variant for task modeling. Examples are Goals, Operators, Methods, and Selection Rules (GOMS) [57], the Task Modelling Language (TaskMODL) [89], and ConcurTaskTrees [68, 70]. In our work, we use a simplified task tree variant that fits our needs. But our task trees may be transformed to the mentioned structures.

There have been attempts to detect tasks in recordings of user actions. A simple detection of tasks means finding a list of actions that are typically performed in a certain combination. Examples for this are the detection of website navigation patterns in Automated Website USability Analysis (AWUSA) [88], Maximal Repeating Patterns (MRP), which provide statistics for action combinations up to a specific length [85, 86], and the detection of repeating action combinations

with the goal to automate them [20]. These approaches detect only rather short reoccurring action combinations and do not generate task models. In our work, we generate full task models for action combinations of any length.

Some other approaches also try to compile full task models based on recorded user actions. The approaches can be distinguished based on certain criteria. At first, there are approaches that generate a probabilistic model for tasks represented in, e.g., Petri nets [18]. These models describe probabilities for action combinations and are derived from statistics. A further variant are approaches that require labeled input data. Labeled herewith means that the beginning and the end of a task execution are marked in a list of recorded actions. The labels are usually defined manually either by the person analyzing the data [29] or by the user when starting a task execution [30] as done in ACT-R [51] and Convenient, Rapid, Interactive Tool for Integrating Quick Usability Evaluations (CRITIQUE) [46]. It is also possible to define them using timestamps [4]. The detection of tasks is then done based on the labels via machine learning [81] or sequence alignment [4]. There is also one approach based on machine learning, that does not require labels [54]. Finally, as Hilbert [44] points out, tasks could be detected using methods for grammatical inference from language examples. This is based on the idea that the interaction between a user and a software can be considered a language spoken by the user and the actions of the user are the words of the language. An example for such an approach is ActionStreams [59]. But these approaches also either require labeled data [25], a manual interaction [66], or do not produce the same result when being filled with the same input data but the data is presented in arbitrary order. The task tree generation used in our approach generates task trees from recorded user actions, does not require labeled data, does not produce only a probabilistic model, and produces the same result independent of the order in which the input data is processed.

3.4 Usability Smell Detection

The major contribution of this article is the identification of usability smells in VR applications. In general, a smell is an occurrence of a certain pattern in some kind of *data* that may indicate the existence of an issue. Hence, a usability smell in some data can be an indicator for a usability issue. Depending on the type of data, there are different definitions of the term usability smell. One of the first works that utilized the term was on static GUI model analysis [3]. Here, a usability smell is considered as some structure in the GUI model that may cause a usability issue during its usage. An example is an unnecessary intermediate dialog on a certain navigation path. Our work does not consider GUI structures but GUI usage as a baseline for usability smell detection.

A further type of data to search for usability smells can be recorded user actions. This is done in the work of Grigera et al. [34, 35]. There, the authors record the usage of individual GUI elements. The corresponding actions are combined to so called usability threats which are higher level actions. These are subsequently analyzed to check if a usability smell is present. This is strongly focused on individual GUI elements or typical GUI structures. For example, they analyze if users often request tooltips for links on a website or if a search result list is empty for a certain search term. A similar but less automated work is the one of Paternò et al. [74]. Here, the authors search for usage patterns, which they also call usability smells and which are defined similar to regular expressions. The drawback of this approach is that it may also find single occurrences of a pattern, i.e., there is no metric indicating relevance. Furthermore, the overall analysis of the findings is again performed manually. Both works do not take into consideration the broader scope of a complete user task as done in our work. Instead, they consider short and predefined action combinations.

The consideration of complete user tasks is the focus when searching for usability smells in task models. Here, a usability smell is a certain structure in a task model that may show a usability issue.

An example for such an approach is GOMS [50]. GOMS can be used to estimate the users efficiency for performing a certain task based on a detailed task model. If the efficiency is considered bad, then the whole task model itself is considered problematic and can be seen as a usability smell. Analyses like GOMS may also take the concrete elements of a user interface and their corresponding types into account [82]. This makes an efficiency estimation more reliable and correct. In addition, by considering such a mapping between a task model and a GUI model, an iterative improvement of both models can be done with the goal of creating a highly efficient task execution [26, 75]. This optimization can also be automated. The drawback of these approaches is that they do not consider the actual usage of the software because the task models are manually defined by a system designer or an evaluator instead of being generated from usage recordings.

To the best of our knowledge, there is yet no related work on automated usability analysis or smell detection for VR. In addition, the existing work does not focus on task trees generated from usage recordings. Hence, the approach proposed in this article is the first in this direction.

4 AUTOMATED USABILITY EVALUATION OF VRs USING GENERATED TASK TREES

In this section, we describe our approach for automated usability evaluation of VRs. This approach consists of the three steps user action recording, task tree generation, and usability smell detection. We implemented this basic concept already for desktop applications and web sites [37–41]. In this article, we transfer this work to VR. We describe the details for the three steps in the following subsections. In these descriptions, we will clarify which parts of our approach were reused from previous work and which parts were changed for their application for VR.

4.1 User Action Recording

The first step of our approach is recording user actions in a VR. As mentioned in the foundations in Section 2, interactive objects and performed actions in a VR differ from those in GUIs. Hence, we could not reuse our corresponding previous work for this step. Furthermore, we did not reuse existing logging concepts, e.g., for game analytics. This allowed us to have more influence on the recorded data. We describe our recording mechanism in the following to make our work reproducible.

Similar to GUIs, actions in a VR cause events in the underlying VR system. Hence, for recording actions in a VR, we intercept the corresponding events and store them as an event list in a human-readable log file. For example, if a user grabs an object in a VR, we record an event named *objectGrabbed* and log it together with an identification of the grabbed object. An example for an event list with two logged events is shown in Listing 1. The first event denotes a grabbing of a target object with the id "cup." The grabbing takes place at the coordinates in the VR denoted by the given target object position. The second event shows a release (*objectUngrabbed*) of the object at a different location.

```
<event type="objectGrabbed">
 <targetId value="cup"/>
 <targetPosition value="(0.12,1.06,0.51)"/>
</event>
<event type="objectUngrabbed">
 <targetId value="cup"/>
 <targetPosition value="(0.78,1.13,0.38)"/>
</event>
```

Listing 1. Example of recorded user actions in form of events of a VR.

Overall, we consider five different events. The first two are the ones mentioned in the example. Two further events are caused by using an object in a VR. Using is different from grabbing as the corresponding object can be interacted with, but its position remains the same. An example is using a light switch. The names of the corresponding events are *objectUsed* and *objectUnused*. The fifth type of events represents head movements. Here, we consider head rotations and directional head movements. To detect head movements, we continuously trace the rotation and the location of the *VR camera*. The VR camera is the view of the user into the virtual world. For state-of-the-art consumer VR systems, the VR camera follows the head movements of the user and, therefore, its position and rotation represents the user's head position and rotation inside the VR. We log a head movement event if either the VR camera position changes with a minimum speed of $v = 0.5$m/s or the VR camera rotates with a minimum angular velocity of $\omega = 20°/s$. In addition, if we observe that the movement direction is not linear, we also log an event for every direction change being larger than 5°. This also means, that slower head movements stay unrecognized in our approach. We decided for these threshold numbers based on experiments in which they showed not to produce too many events, but still all significant head movements are recognized.

The event logging requires an adaption of the VR internal event handling, i.e., a corresponding extension of the relevant source code. Doing this manually would require a lot of effort. Hence, we considered an automation for this extension. This automation works as follows. Similar to websites, which have an underlying document object model, virtual worlds are defined through a directed graph, the *scene graph* [24], whose nodes, in addition to others, represent all the visible and interactive objects of the VR. State-of-the-art game engines, e.g., Unity 3-D [91], which can be used for consumer VR creation, allow to programmatically traverse the scene graph and to access and change its nodes. Nodes representing interactive objects in a VR have certain characteristics that identify them as interactive. For example, in Unity 3-D, these nodes have certain scripts attached that take over the corresponding event handling. Through these attached scripts, we are able to identify interactive objects. In addition, these scripts offer possibilities to programmatically analyze and extend the already attached event handling with further functionality. We use this possibility to add the logging of events to the existing event handling. For this, we require information, e.g., an identification of the interactive object and the type of event. This information can be retrieved from the scene graph and the event handling scripts, too.

Using this as a basis, our approach traverses the scene graph for interactive objects at start up of a VR and extends their event handling with a logging for each event. Then, if a user triggers an event with a certain action, the event is handled by the VR system and additionally logged. As during the execution of the VR, the scene graph may change, i.e., new interactive objects may be added and others may be removed, our approach continuously checks the scene graph for modifications and, if required, performs changes so that the correct logging of events is ensured. Through this basic concept, it is rather simple to add the event logging mechanism to an existing VR.

For the logging itself, we do not store the resulting events locally, but send them to a central server. This server combines them to event lists and persists them. This allows for an easier aggregation of all recorded data and a centralized subsequent processing, e.g., for the task tree generation.

4.2 Task Tree Generation

The next step of our approach is the generation of task trees from the event lists that result from our VR user action recording. For this, we reuse and extend an algorithm that we developed in previous work [37, 40, 41]. This algorithm processes the recorded event lists. It works by iteratively detecting iterations and sequences. For a single event list, this is visualized in Figure 2. Figure 2(a) shows an event list where each event is a box with a character. Identical events, i.e., user actions,

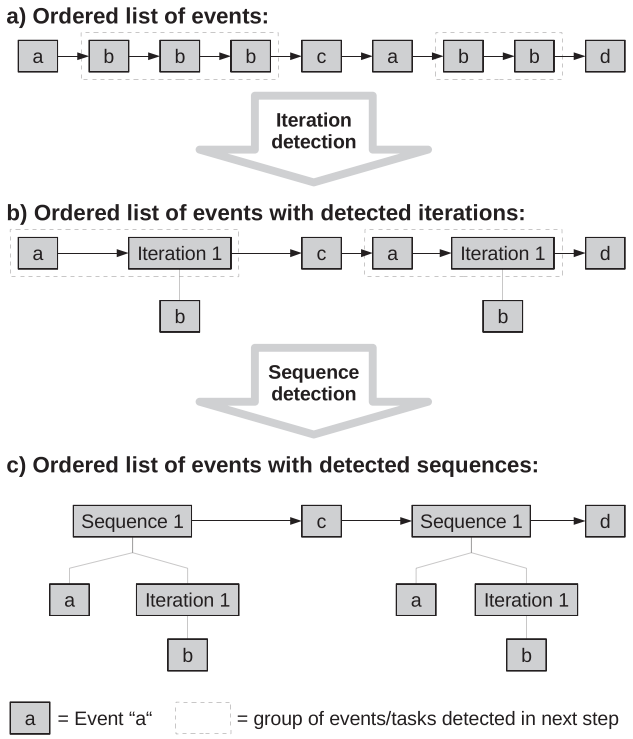

Fig. 2. Example for generating task trees as proposed by Harms et al.

have the same character. At first, the algorithm searches for subsequent identical events, as for example event *b*. If there are such events, the algorithm replaces them with an iteration of the event as visualized with *Iteration 1* in Figure 2(b). In the next step, the algorithm searches for sublists occurring multiple times. If there are any, as the sublist *<a, Iteration 1>* in the example, they are replaced with a corresponding sequence (*Sequence 1* in Figure 2(c)).

After the sequence detection, the algorithm starts a further iteration detection to detect subsequent identical sequences. Then, it continues with a further sequence detection. This alternating iteration/sequence detection is continued until no further iterations or sequences are detected.

The task tree generation can be applied on multiple event lists at the same time. In this case, the sequence detection works on multiple event lists. When searching for sublists occurring multiple times, the sequence detection also considers multiple occurrences in all event lists. This means, even if a sublist occurs at most once in an individual event list, it may occur multiple times considering all event lists and will, hence, be replaced by a sequence in all event lists.

During a sequence detection, usually multiple sublists are detected that occur multiple times. But the algorithm considers only one of them during one sequence detection cycle. For the selection, which sublist is preferred, the algorithm defines a detailed mechanism consisting of several selection criteria. The major selection criteria focuses on the number of times a sublist occurs and prefers the sublist occurring most often. Further selection criteria are described in [37].

The task trees resulting from the algorithm have diverse important characteristics. First of all, not all events may be considered in the task tree generation. In Figure 2, this is, e.g., the case for event *c* and *d*. This happens, if certain event/event or event/task combinations occur only once in the recordings. This in turn means, that all detected task trees do not necessarily represent and,

hence, cover all recorded events. On the other side, the task trees generated with this approach have shown to be representative for the recorded user behavior [37, 39]. This means, they correctly model the usage of the software. This representativeness typically varies for the different detected sequences. Analyses have shown that those 20% of all detected sequences that are based on sublists that occurred most often already describe 80% or more of the recorded user actions, i.e., events (Pareto principle) [37]. Those sequences are the so called *prominent sequences*. The other sequences, as well as the remaining events, are less representative and mainly describe noise in the data, i.e., action combinations that occurred rather seldom.

A further important characteristic is that the more usage data is recorded, the more representative are the task trees [37]. Furthermore, when generated on recordings of multiple users, the task trees represent usage commonalities for different users. In addition, the more consistent the users behave, the less complex and the more representative are the resulting task trees. Overall, these characteristics make the task trees a good source for a detailed analysis of the software usage.

Due to the basic concept of their generation, the task trees allow for a back referencing to all information from which they were generated. This includes the individual events, their types, and number of their occurrences. Furthermore, all details stored in the recordings, such as coordinates of objects with which the users interacted, can be traced back, too.

The major change we made to the algorithm for applying it on VR event lists is based on the fact that the algorithm needs to decide which events are identical and which not. The existing implementation was able to compare user events on website and Java applications. We additionally enabled it to compare VR events. For this, we defined the following identity criteria. We consider two events as identical if they

—are of the same type and
—refer to the same interactive object.

This means, e.g., two *objectGrabbed* events on the same target object are considered identical. In contrast, two *objectGrabbed* events on two different target objects are considered distinct.

4.3 Usability Smell Detection

Based on the generated task trees, we perform an automated usability evaluation using the detection of usability smells. We consider usability smells to be certain user behavior that indicates an underlying usability issue. In previous work, we showed a respective approach for websites and desktop applications [37, 38]. There, we searched for six usability smells in generated task trees [37, 38]. Two of them have a strong focus on traditional GUIs and are, hence, not considered in this article. The first of the other four is called *Important Task*. It searches for those detected sequences that were executed most often, respectively, which cover most of the recorded events. The second smell is called *Required Inefficient Actions*. It analyzes task trees for inefficient actions such as scrolling on websites. The third smell is called *High GUI Element Distance*. It determines the average distance between two subsequently used GUI elements, which should not be too high to ensure a good user efficiency. The fourth smell is called *Missing User Guidance* and does not search for a specific task pattern, but analyzes the ratio of recorded actions that belong to a detected task. The higher this ratio, the more consistent the users behave and the more they are guided by the user interface. These usability smells have proven to provide valid findings for websites and desktop software and, therefore, to contribute to a usability engineering process [37, 38].

For applying the usability smell detection on VR, we reused two of the above-mentioned usability smells, adapted the two others, and defined a new one. The two reused usability smells are *Important Tasks* and *Missing User Guidance*. The adapted smells are *Required Inefficient Actions* and *High GUI Element Distance*. As the name of the latter one includes the term GUI, which does not

match for VR anymore, we also renamed this smell to *High Interaction Distance*. The newly defined smell is called *Task Retried*. It considers repetitions of complete tasks.

We selected these five smells for multiple reasons. First of all, they can be detected based on the generated task trees. We did not consider other smells, e.g., smells that can be detected on pure recorded events as done in our previous work. This would have made the scope of the paper too large. In addition, we think that the selected smells can be indicators for a broad variety of concrete usability issues and can, therefore, contribute well to a usability engineering process. Finally, the detection of the smells is easy to implement and simple to reproduce by other researchers.

We describe all smells in detail in the following. For each smell, we provide a reason for inclusion. This is a reference to corresponding literature, e.g., guidelines and standards, as argument for a smell. Afterwards, we provide for each smell the expected user behavior as it should occur in the task trees if the related usability issues are present. Then, we give a description of the smell detection. This includes the calculation of an *intensity*, which is a measure of how likely a finding for a smell indicates a usability issue. Subsequently, we describe the possible usability critique the findings for a smell can provide. Finally, we describe the system message as well as additional information that our approach provides for each smell finding. This message includes details on how to solve the underlying usability issue of a finding. The additional information are, e.g., visualizations of the relevant task tree or the relevant interactive objects in the scene graph of the analyzed VR.

US1 - Important Tasks

Reason for Inclusion (Guideline Reference)

User tasks that are executed often, and which are, hence, important, should be executable with a high efficiency [1, 65]. This can be achieved by reducing the number of required actions [56, 87, 90] or by automation [86].

Expected User Behavior in Task Trees

Users will execute important tasks more often than others and always in a similar fashion. Hence, representations of these tasks should be found in the generated task trees.

Detection and Intensity

The detection of this smell focuses on sequences only. For each sequence s, we determine the number of events $e(s)$ from which s was detected. We normalize this to the number of all recorded events $|E|$ by calculating:

$$I_{\text{impTask}}(s) = \frac{e(s)}{|E|}$$

$I_{\text{impTask}}(s)$ is then the ratio of all recorded events which represent executions of s. This is also the intensity of the findings for this smell. The higher this ratio, the more important is s. The smell detection ignores findings with an intensity below 0.001, i.e., findings for sequences that represent less than 0.1% of the recorded data. The reason for this is that our previous work [37, 38] shows that tasks with a rather low event coverage are also less representative for the users' tasks and that they represent noise instead.

Usability Critique

Based on their intensity, our approach orders findings of this smell by their relevance starting with the highest intensity. This allows the smell to identify those sequences, and, hence, tasks, being of most importance for the users and to propose to minimize the required actions for performing the task.

System Message For each detected sequence s, for which $I_{\text{impTask}}(s) \geq 0.001$, our approach provides the following message:

> "The task <task ID> covers <ratio> % of all recorded user actions. Therefore, it should be executed with a high efficiency. You should consider making the task more efficient, by minimizing the number of actions to be executed by users for fulfilling this task."

where <task ID> is the id of s and <ratio> is $I_{\text{impTask}}(s)$ given in %.

Additional Information A finding for a sequence s additionally provides a visualization of the task tree of s as well as an excerpt of the scene graph with all interactive objects of the analyzed VR that were utilized while executing s.

US2 - Required Inefficient Actions

Reason for Inclusion (Guideline Reference) Inefficient actions have no semantic effect on the task fulfillment [71], i.e., they do not contribute to the logical completion of the task, but only cost time. Hence, they should be minimized for increased user efficiency [65].

Expected User Behavior in Task Trees If certain inefficient actions must be performed for completing a certain task, users will perform them and the actions will be included in the generated task trees.

Detection and Intensity The detection of this smell focuses on sequences only. For each execution s' of all executions $x(s)$ of a sequence s, we first determine the number of inefficient actions $ia(s')$. As inefficient actions, we consider head movements and rotations. Then, we also determine the number of effective actions $ea(s')$ performed in the execution s' of s. Finally, we calculate the average ratio of inefficient actions typically performed when executing s, which is also the intensity of the smell, as follows:

$$I_{\text{ineffAct}}(s) = \frac{1}{|x(s)|} \sum_{s' \in x(s)} \frac{ia(s')}{ea(s') + ia(s')}$$

Usability Critique Based on the intensity of the findings for this smell, our approach identifies and orders the sequences, i.e., tasks, for which a high number of inefficient actions are executed. It then proposes to restructure the tasks so that the inefficient actions, i.e., head movements and rotations, are not required or at least minimized.

Head movements and rotations may not always be inefficient. For example, for exploring a VR they are important. We expect that users execute such tasks less often than tasks for which efficiency is more important. For example, users repeat the exploration of a VR only if necessary. In contrast, users do mandatory tasks, e.g., the usage of a menu, as often as required. Hence, we expect that based on the number of executions of tasks for which this smell is found we should be able to separate infrequent exploration tasks from more frequent tasks for which this smell is of importance.

System Message For each detected sequence s our approach provides the following message:

> "For executing task <task ID>, the user has to execute a high number of inefficient actions (<ratio> % on average). Such actions are, e.g., scrolls or head movements. The number of inefficient actions should be as minimal as possible to increase the user's efficiency in executing a task. Please check the task structure for occurrences of the following event tasks and consider to prevent them: scrolling (usually prevented by matching the size of a view to the available screen space), head movements (usually prevented by putting interactive objects closer together."

where the <task ID> is the id of s and <ratio> is $I_{\text{ineffAct}}(s)$ given in %.

Additional Information A finding for a sequence s additionally provides a visualization of the task tree of s as well as an excerpt of the scene graph with all interactive objects of the analyzed VR that were utilized while executing s.

US3 - High Interaction Distance

Reason for Inclusion (Guideline Reference) To achieve an efficient task execution, the user interface of a software, e.g., a VR, should be structured in a way, so that no long distances must be overcome by users [56, 65, 84].

Expected User Behavior in Task Trees Users will interact with objects in a VR as required for their task execution. The generated task trees represent this task execution and their leaf nodes reference the performed user actions. These actions in turn refer to the respective target objects and their locations in the three-dimensional VR space.

Detection and Intensity The detection of this smell focuses on sequences only. For a sequence s, we first determine all effective actions and the corresponding VR objects that the users interacted with in any execution of s. Then, we determine the coordinates of these objects in the three-dimensional space by using the trace back mechanism from the task trees to the recorded events. Using the euclidian distance in the three-dimensional space, we then calculate the distance between any subsequent pair of used interactive objects. This results in a set of distances $D(s) = d_1 \dots d_n$. Using this set, we determine the average distance between two subsequently used objects in executions of a sequence s which is also the following intensity of the smell:

$$I_{\text{highDist}}(s) = \frac{\sum_{d \in D} d}{|D|}$$

Usability Critique Based on the intensity of the findings for this smell, our approach identifies and orders the sequences, i.e., tasks, for which users of the VR have to travel high physical distances. It then proposes to restructure the VR so that smaller distances must be overcome, e.g., by putting subsequently used objects closer together.

System Message For each detected sequence s our approach provides the following message:

> "For executing the task <task ID>, the user utilizes <no of interactive objects> objects on average. As these objects have a high distance to each other (<average distance> on average), they should be more co-located to ease the execution of the task for the user."

where <task ID> is the id of s, <no of interactive objects> the average number of interactive objects used when executing s, and <average distance> is $I_{\text{highDist}}(s)$ given in the unit used by the VR internal coordinate system. (For Unity 3D this unit is meters).

Additional Information A finding for a sequence s additionally provides a visualization of the task tree of s as well as an excerpt of the scene graph with all interactive objects of the analyzed VR that were utilized while executing s.

US4 - Task Retried

Reason for Inclusion (Guideline Reference) If users make errors, a system shall allow for their correction [63]. In addition, if errors are likely at a certain point of interaction, then the system shall implement error prevention [63]. For this, a system shall detect user errors [65].

Expected User Behavior in Task Trees If users make the same errors and the same means of corrections multiple times in a row, this will occur in the generated task trees as repeated tasks, i.e., iterations of sequences.

Detection and Intensity For the detection of this smell, we search for iterations of sequences. For any repeated sequence s, we determine the number $r(s)$ of direct repetitions of s after an initial execution of s. Based on this and the number of effective actions $ea(s)$ belonging to s, we calculate the following measure which serves as the intensity of this smell:

$$I_{\text{taskRetry}}(s) = r(s) * ea(s)$$

Usability Critique For each finding of this smell, our approach provides the repeated task as well as the number of its repetitions. In addition, it proposes to prevent unnecessary task repetitions. The intensity of the findings is used for sorting them with respect to their likelihood of representing an issue.

System Message For each detected sequence s our approach provides the following message:

> "The task <task ID> is often retried, i.e., repeated subsequently. On average, every <count>$^{\text{th}}$ instance is followed by one or more repetitions of the task and the average repetition ratio when a task is repeated is <ratio> additional repetitions. This can be task retries and may indicate that the task is hard to be performed. Hence, it should be eased up. This can be done by checking why the task is retried and then adapting the implementation so that the task can be finalized with its initial execution."

where <task ID> is the id of s, <count> represents the ratio of initial executions of s which are followed by at least one repetition, and <ratio> is the average number of repetitions of s in case it is repeated.

Additional Information A finding for a sequence s additionally provides a visualization of the task tree of s as well as an excerpt of the scene graph with all interactive objects of the analyzed VR that were utilized while executing s.

US5 - Missing User Guidance

Reason for Inclusion (Guideline Reference) When using a software, users require guidance [63]. Otherwise, they will act inefficiently [9, 10, 56].

Expected User Behavior in Task Trees The more guidance the users have, the more they perform the same order of actions. Considering the task tree generation, this should result in few detected tasks covering most recorded actions. If no user guidance is present, then only a few tasks will be detected.

Detection and Intensity For the detection of this smell, we consider all recorded events and determine for every 10 subsequent events the ratio a_i^{10} of tasks that were detected on them. If an event list is shorter than 10 events, it is ignored. This results in a list of ratios $A^{10} = a_1^{10} \dots a_n^{10}$ for each list of 10 subsequent events. From this list, we determine the average ratio which results in the following calculation of the smell's intensity:

$$I_{\text{guidance}} = \frac{1}{|A^{10}|} \sum_{a_i^{10} \in A^{10}} a_i^{10}$$

We do this calculation twice. Once we determine the ratio considering all detected tasks, and once considering only the prominent sequences. Hence, this smell always has two findings, one for the overall ratio and one for the ratio considering only prominent sequences.

Usability Critique Without a reference to an individual task, this smell is an overall measure for the user guidance. Therefore, it cannot provide more details on how the issue can be solved.

System Message For each finding of this smell our approach provides the following message:

> "The user sessions show only a few commonalities. On average, ten subsequently executed actions need to be described by <ratio> different determined tasks. In the worst case, each action is its own task. In the best case, all actions are described by the same tasks. This indicates, that the users act relatively different as otherwise, all their actions would be described by only a few tasks. Hence, users seem to be missing guidance, as otherwise, they would behave similar resulting in less generated tasks which still cover many recorded actions."

where the <ratio> is I_{guidance}.

Additional Information A finding for this smell does not provide further information.

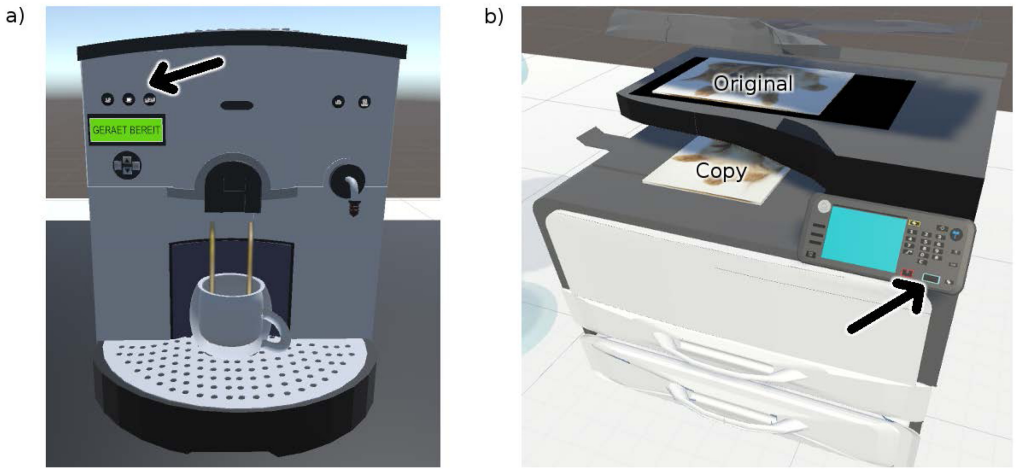

Fig. 3. Screenshots of the devices used in both scenes of our case study. Left brewing a coffee with a virtual coffee machine, right copying a paper using a copier. The buttons marked by the arrows must be pressed to brew a coffee or to copy a sheet of paper.

We are aware, that these usability smells may not cover all usability aspects. They will not be able to identify certain usability issues, e.g., user misunderstandings. This can also be expected by the fact, that we do not reference corresponding guidelines in the reasons for inclusion. But with their focus on user efficiency and user errors, they may help to identify issues related to these aspects. In our case study, which we describe in the following, we assessed the capabilities of our approach in detail.

5 CASE STUDY

To evaluate our approach and to answer our research question, we performed a case study with two different VR *scenes*. A scene is a virtual room with a dedicated setup. In one of the scenes, the users' task was to brew a coffee with a virtual coffee machine. Hence, we call this scene *coffee scene*. A screenshot of the coffee machine is shown in Figure 3(a). In this scene, the users first had to grab a virtual cup, move it a certain distance, and then put it under the coffee outlet in the center of the machine (already done in the figure). Then, the users had to initiate the brewing by pressing one of the coffee machine buttons which, to their opinion, was the one that fitted their goal most. If the users correctly pressed one of the three buttons on the top left of the coffee machine (see arrow in the figure), the coffee started to flow.

In the other scene of the case study, the users were asked to copy a sheet of paper using the virtual copier in Figure 3(b). We named this scene *copier scene*. In the scene, the users had to open the copier top, put a sheet of paper onto the copier, and press the button of the copier that seemed to match their goal. Also here, the paper had to be moved via a certain distance. If the correct button (see arrow in the figure) was pressed, then a virtual copy of the paper appeared in the outlet of the copier. Figure 3(b) shows the copier in the state after all the actions have been performed. Therefore, it shows two sheets of paper, the original and a copy.

We selected these two scenes for several reasons. First of all, the tasks are not too complex making it easier to introduce them to participants. In addition, the actions of grabbing, moving, and using objects are the most used actions for VR when interacting with interactive objects. The only action that we do not cover explicitly is locomotion. In state-of-the-art VR, locomotion is done in different ways. For example, it is possible to move oneself in the tracking area or use some

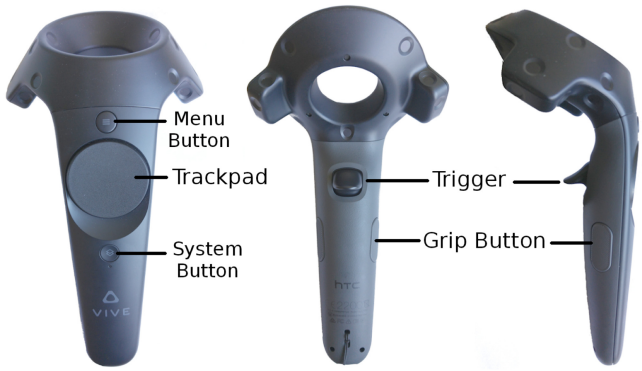

Fig. 4. Visualization of an HTC Vive controller with its buttons.

kind of joystick. Many VRs use a teleporting mechanism, where the users point at a location in the VR and then teleport to it by pressing a controller button. Our scenes are large enough, so that a certain movement of the users in the tracking area is possible and partially required to interact with the objects. Therefore, we cover the first technique for locomotion. We did not implement the other techniques to not make our users' task too complex and to have more control over the experiment. But as our approach also tracks the users' head position, which changes independently of the concrete locomotion technique, we expect the concrete locomotion technique not to be of too much influence for our analyses. Overall, the implementation of own VR scenes allows us to better control our experiment and to evaluate different interaction modes, which are described in the following.

5.1 Technical Setup and VR Implementation Details

For the technical setup of the case study, we used an HTC Vive (Vive) as VR device [19]. The Vive comes with a head-mounted display and two controllers all being tracked at room scale. A visualization of a Vive controller is shown in Figure 4. Per default, the controllers have a virtual representation of their real form in the VR. This means, when moving a controller in real space, its virtual representation in VR is also moved accordingly.

The controllers have several buttons marked in Figure 4. In our case study, we only used the trigger button at the bottom and the track pad at the top of the controller. Additionally, we put a function for resetting the scene on the grip buttons of the controller. This restores the original position of the cup in the coffee scene or the paper and the copier in the copier scene. For the case study, at most one controller is used at a time.

The scenes of the case study were utilized with four different interaction modes. Each interaction mode defines in which way the headset and a controller are used to perform certain interactions like grabbing a paper. The interaction modes focus on different technical levels of VR systems with which they can be implemented. We considered these modes to validate if, with respect to a better answering of our research questions, our approach can be applied to different interaction techniques and technical setups for VRs. This is required, as nowadays, different technical setups are available and already spread in the market. The basic concepts of the interaction modes and the targeted VR systems are as follows:

— The first interaction mode works only with a gaze pointer and without a controller. Therefore, we call it the *gaze mode*. To interact with an object in a VR, the users only have to look at it by pointing a crosshair in the center of their view over the object. This mode can be

implemented with Google Cardboard [93] like state-of-the-art VR systems. These systems typically only provide head tracking and no controller support for user interaction.

— The second interaction mode utilizes one controller that works as a virtual laser pointer. Therefore, the mode is called the *laser mode*. In this mode, a laser beam radiates from the virtual controller representation. To interact with an object in a VR, the users need to point at it with the laser beam and press the trigger button on the controller. This interaction mode focuses on VR systems with minimal tracked controller support as, e.g., Google Daydream [94].

— The third interaction mode, called the *controller mode*, also utilizes a controller, but distinguishes between the actions of grabbing an object and pushing a button on the coffee machine or the copier. For grabbing an object, users have two options. Either they collide the virtual controller representation with the object or they can point at it with a laser beam. In contrast to the laser mode, the laser beam is not emitted continuously but only if the trigger button is slightly pulled. For actually triggering the grab action, in both variants the users, additionally, have to push the track pad on the controller. For pushing a button on the coffee machine or the copier in this mode, the users need to point at the button with the laser beam and afterwards, fully pull the trigger so that a click in the controller is observed. This mode can be implemented with high quality state-of-the-art VR systems, like Oculus Rift [64], HTC Vive [19], or Microsoft Mixed Reality devices [61].

— The last interaction mode uses no controllers but an additional device for tracking the users' hands. Therefore, this mode is called the *hand mode*. In this mode, the users were able to see and utilize their hands in the VR scenes for grabbing and using objects. We implemented this mode using the Leap Motion [55]. The Leap Motion can be attached to the VR headset and is then able to track the hands of the users. We used the Leap Motion Interaction Engine [12, 55] to show a virtual representation of the users' hands in the VR. This interaction mode can also be implemented with other techniques, e.g., VR gloves [95].

More details on the four interaction modes for interacting with the VR are shown in Table 1. This table consists of three columns. The first lists the actions that are of interest in our case study, like grabbing an object or pushing a button on our virtual devices. The second column lists the different interaction modes, e.g., the gaze mode. The third column describes per action and mode, how the action is triggered also referencing the buttons of the Vive controllers, if required.

Three of the interaction modes come with feedback for users. For the gaze mode, every interactive object in the VR scene changes its color to green if the crosshair is placed on it. The same applies if a user points at an interactive object with the laser beam in the laser and controller mode except that the highlighting color is red. In the hand mode, no highlighting is implemented. Because users have to wait a second until an action on an interactive object is triggered in the gaze mode, we implemented a color blur from green to red in this mode. When the full red color is reached, the object is grabbed or used, respectively.

5.2 Data Recording

Overall, we asked 84 persons to use the VR scenes. We did this in two separate sessions in June 2017, of which one took place at our university (44 participants) and the other in a shopping mall (40 participants). This is also the reason for the overall high number of students (45) in contrast to other persons. The participants were of different ages ranging from teenagers to retired persons. The previous experience of the participants with VR ranged from no experience (65 participants) via heaving heard about it (14 participant) to being already familiar with it (4 participants). None of our participants was an expert in VR. Further details on the participant data are available in the provided replication kit (see Section 7).

Table 1. Different Interaction Modes Used in the Case Study and the Description of How the Different Interactions with the VR are Triggered

	Mode	Description
Grab object	Gaze	Move crosshair in view center on interactive object for a second
	Laser	Point with laser pointer on interactive object and pull trigger on controller
	Controller	Collide controller with interactive object or slightly pull trigger on controller to activate laser pointer and point with it on tangible item, then press the track pad
	Hand	Close fingers around interactive object
Move object	Gaze	Grab object and afterwards move the head
	Laser	Grab object and afterwards move the controller
	Controller	Grab object and afterwards move the controller
	Hand	Grab object and afterwards move the hand
Drop object	Gaze	Automatically if destination or origin is reached
	Laser	Release the trigger on the controller
	Controller	Release the track pad on controller
	Hand	Open fingers
Use button	Gaze	Move the crosshair in view center on the button for a second
	Laser	Point with the laser pointer on the button, afterwards pull and release the trigger on the controller
	Controller	Slightly pull the trigger on the controller to activate the laser pointer, then point on the button with laser pointer, and fully pull the trigger
	Hand	Push the button using the tip of any finger

Every participant used both scenes but only one interaction mode (22 gaze, 21 laser, 21 controller, and 20 hand). The ordering of the VR scenes was changed so that 44 participants first used the coffee machine and the others first used the copier. On average, the participants spent 63 seconds in the coffee scene and 87 seconds in the copier scene.

At the beginning of each individual session, we thanked the respective participant for the support and informed about the reason and the setup of the case study. We also informed about the logging mechanisms taking place in the background and that the data are stored anonymously. Afterwards, we let the participant put on the head-mounted display. In case of a controller-based mode, we also introduced the controller buttons. Then, we started the first of our scenes. For the scenes, we only formulated the task but not, how it is solved. Most importantly, we did not describe how the interaction modes work.

For recording the user actions in our case study, we extended the VRs implementation with a mechanism that traces the user actions and sends them to a server for logging as described in Section 4.1. This server and any subsequent processing step in the case study was set up using the Automatic Quality Engineering of Event-driven Software (AutoQUEST) framework [28]. This

Table 2. Recorded Events and Generated Task Trees for the Two Different VR Scenes and the Four Interaction Modes

	Coffee scene				**Copier scene**			
	Gaze	*Laser*	*Controller*	*Hand*	*Gaze*	*Laser*	*Controller*	*Hand*
Sessions	22	21	21	20	22	22	21	21
Events	1063	1028	1004	2339	2000	1557	1979	4364
Actions	17	21	15	23	21	17	19	59
Session length μ	48	49	48	117	91	71	94	208
Session length σ	17.1	18.2	20.6	92.5	33.8	25.8	38.1	97.5
Sequences	28	23	14	83	40	31	42	134
Iterations	10	7	4	40	11	11	10	55
Coverage	96%	97%	98%	90%	96%	97%	98%	88%
Prominent Seq.	6	5	3	17	8	7	9	27
Ratio	21%	22%	21%	21%	20%	23%	21%	20%
Coverage	75%	81%	75%	61%	70%	77%	82%	74%

framework already comes with a server for storing logged user actions and a subsequent workflow for their processing. In addition, AutoQUEST implements the task tree generation approach described in Section 4.2. Hence, we mainly had to implement the required identification of identical user actions and the usability smell detection described in Section 4.3.

For all users, we recorded the five different event types mentioned in Section 4.1. This resulted in 15,334 recorded events. The distribution of these events on the two VR scenes as well as the interaction modes is shown in Table 2. In this table, each column denotes a combination of VR scene and interaction mode. The first row contains the number of sessions recorded for each scene/mode combination. This number correlates with the above-mentioned distribution of the modes on the participants except for the hand and laser mode for the copier scene. Here, the number of recorded sessions is higher by one because of two session interruptions and restarts due to technical issues with the Vive headset. Anyway, we considered the incomplete sessions, as they already included parts of the interactions of the users with the scenes. The second row of the table contains the number of events recorded for each scene/mode combination, while the third row shows the different types of actions. Considering that for all modes in the respective scenes the task was the same, it is interesting to note that the number of recorded events for the hand mode is about twice as high for both scenes than for the other modes. Also, the average values and the standard deviations of the session lengths for the modes, which are listed in lines four and five of Table 2, support this difference. We performed a pairwise two-sided Wilcoxon test [98] on the session lengths with a Hochberg p-value adjustment [45] for both scenes to check if this difference is significant. The null hypothesis that the average session lengths are equal between the respective pairs is rejected for all pairs, including the hand mode (p-values below 0.005 [11]), which underpins the session length difference.

5.3 Task Tree Generation

Based on this recorded data, we performed the task tree generation described in Section 4.2. The resulting numbers of sequences and iterations are listed in the second part of Table 2. The Spearman's rank correlation coefficient [99] with a value of 0.966 shows a high potential for a correlation between the number of events and the number of generated sequences. The coverage in this part of the table shows the percentage of input events, which are described by the generated sequences.

All values except for the hand mode in the copier scene are above 90%. The number of prominent sequences and the percentage of recorded events they cover are listed in the last part of Table 2. Also here, the coverage is rather high with at least 70% except for the hand mode in the coffee scene.

In the table, we also provide a ratio for the prominent sequences in the second to last row. Although mentioned in Section 4.2 that the prominent sequences are 20% of all generated sequences, this is not always exactly the case. In our case study, the reason is that the amount of all generated sequences may not be dividable by five to be able to exactly create a subset with 20% of all generated sequences. In addition, the prominent sequences are those covering most of the recorded actions. But there may be multiple sequences, that cover the same amount of recorded actions. Hence, when filling the subset of 20% of those sequences covering most of the recorded actions, we may have to decide between multiple sequence which to include and which not. To overcome this, we perform the following steps to create the subset of the prominent sequences, which we also already took in our previous work [37, 39]: We first divide all generated sequences S based on the amount of recorded actions from which they were created and create subsets $S'_1 \dots S'_n \subset S$ where $\forall s \in S'_i : s \notin S'_j$ and where each $s \in S'_i$ is generated from, i.e., covers, the same amount of recorded events. Then, we create an empty set S_p for the prominent sequences. Afterwards, we join S_p with sets $S'_i \in S'_1 \dots S'_n$ where S'_i always contains the sequences with the highest coverage. In the same step, we remove S'_i from $S'_1 \dots S'_n$. After each join, we check if $|S_p| > 0.2|S|$, i.e., if it contains at least 20% of all detected sequences. If this is achieved, we stop the joining and S_p contains the prominent sequences. Through this, the last joined set S'_i may cause that the prominent sequences become more than 20%. But through this approach, we ensure a deterministic selection of the prominent sequences.

5.4 Usability Smell Detection

Based on the detected task trees, we performed our usability smell detection as described in Section 4.3. The resulting number of findings are listed in Table 3. For each task related smell, there is a table section of five rows. The first row lists the number of findings in all detected tasks, the second the number of findings in the prominent tasks only. The other rows list the number of findings related to certain activities in the VR scenes like grabbing and moving the cup or using the right button. The sum of the findings in these rows is not necessarily the number of all findings as a finding may refer to more than one activity. For the non-task related smell US5 – Missing User Guidance, there are always exactly two findings due to its nature. Therefore, we listed not the number of findings for the smell but the intensities of the two findings. The first is the intensity considering all tasks, the second considering the prominent tasks only. The table has two column sections, one for the coffee scene and one for the copier scene, both being subdivided into two columns per interaction mode. The first column lists the number of findings for a specific smell for the respective interaction mode and VR scene. The second column, in which the numbers are in braces, lists the number of so called *duplicate* findings. A duplicate is a finding for a certain task which is a direct or indirect child of another task for which there is also a finding for the same smell. Such a separation is not done for US5 – Missing User Guidance as the findings have no task relation. As an example for reading the whole table: for the coffee scene in the laser mode, there are 14 findings for US2 – Required Inefficient Actions of which seven are duplicates. Four findings with two duplicates refer to prominent tasks. Eleven findings with five duplicates are related to grabbing the cup. In this scene/mode combination, the intensity of US5 – Missing User Guidance is 1.5, indicating that on average 10 subsequently recorded actions are represented by 1.5 detected tasks.

Manual Inspection of the Findings. We did a manual inspection of all findings to evaluate if they correctly indicate usability issues. For these inspections, we first visualized the findings.

Table 3. Findings for the Usability Smells Separated by the Two Different VR Scenes and the Four Interaction Modes

		Coffee scene								**Copier scene**							
		Gaze		*Laser*		*Contr.*		*Hand*		*Gaze*		*Laser*		*Contr.*		*Hand*	
		Count	Dupl.	Count	Dupl.	Count	Dupl.	Count	Dupl.	Count	Dupl.	Count	Dupl.	Count	Dupl.	Count	Dupl.
US1 - Important Tasks	Findings	28	(16)	23	(10)	14	(8)	83	(37)	40	(23)	31	(20)	42	(24)	109	(37)
	Prominent	6	(3)	5	(3)	3	(2)	17	(10)	8	(6)	7	(4)	9	(6)	27	(15)
	Grabbing cup/paper	17	(11)	13	(7)	9	(4)	17	(7)	21	(11)	18	(10)	24	(12)	26	(13)
	Right Button Usage	10	(3)	8	(4)	6	(4)	26	(10)	9	(4)	8	(4)	8	(3)	20	(8)
	Wrong Button Usage	6	(2)	5	(0)	2	(0)	42	(18)	6	(3)	6	(3)	3	(1)	72	(17)
US2 - Required Inefficient Actions	Findings	19	(10)	14	(7)	8	(3)	28	(8)	31	(14)	25	(15)	33	(16)	35	(14)
	Prominent	5	(2)	4	(2)	1	(0)	13	(6)	6	(4)	7	(4)	6	(3)	22	(10)
	Grabbing cup/paper	14	(8)	11	(5)	6	(1)	15	(5)	19	(9)	17	(9)	22	(10)	21	(9)
	Right Button Usage	7	(1)	6	(3)	4	(2)	8	(1)	8	(3)	7	(3)	7	(2)	5	(2)
	Wrong Button Usage	3	(1)	0	(0)	1	(0)	5	(1)	3	(0)	3	(1)	1	(0)	6	(1)
US3 - High Interaction Distance	Findings	16	(9)	11	(5)	6	(1)	37	(11)	29	(15)	23	(14)	30	(14)	87	(23)
	Prominent	4	(2)	3	(1)	1	(0)	11	(6)	5	(4)	5	(3)	6	(3)	23	(12)
	Grabbing cup/paper	15	(9)	11	(5)	6	(1)	16	(6)	20	(10)	18	(10)	22	(10)	23	(10)
	Right Button Usage	5	(0)	3	(1)	2	(0)	13	(4)	6	(2)	5	(1)	4	(0)	15	(4)
	Wrong Button Usage	1	(0)	0	(0)	1	(0)	13	(1)	0	(0)	1	(0)	0	(0)	57	(9)
US4 - Task Retried	Findings	8	(2)	6	(0)	3	(0)	17	(7)	9	(3)	10	(5)	8	(1)	17	(4)
	Prominent	1	(0)	2	(0)	0	(0)	8	(5)	3	(2)	4	(2)	2	(1)	7	(3)
	Grabbing cup/paper	4	(1)	2	(0)	1	(0)	5	(3)	5	(2)	3	(1)	5	(1)	6	(3)
	Right Button Usage	2	(0)	0	(0)	1	(0)	5	(2)	0	(0)	2	(1)	1	(0)	3	(0)
	Wrong Button Usage	2	(1)	4	(0)	1	(0)	4	(0)	2	(0)	2	(1)	0	(0)	7	(0)
US5 - Missing User Guidance	Intensity	1.4		1.5		1.2		2.1		1.4		1.5		1.4		2.4	
	Prominent intensity	1.6		1.8		1.3		2.3		1.9		1.7		1.7		2.5	

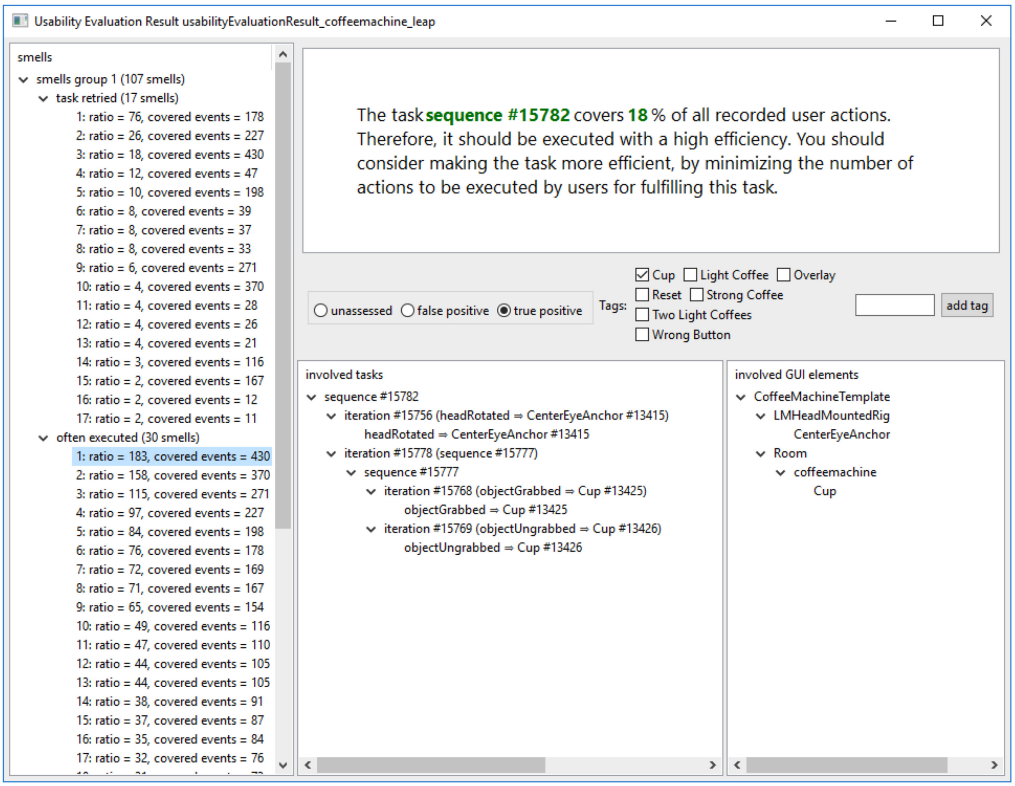

Fig. 5. Visualization of a finding for US1 – Important Tasks as shown by AutoQUEST.

An example for a visualization of the findings for the hand mode in the coffee scene is shown in Figure 5. This visualization was implemented in AutoQUEST. On the left side, there is a tree structuring the findings. On the top level, the findings are grouped to have at most 30 findings per smell type in one of the groups. The groups are created based on the tasks which are referenced by the findings and their event coverage. Hence, the first group contains all findings for tasks that have the highest event coverage. On the second level, the findings are grouped by smell type.[1] The leaf nodes are the respective findings ordered by their intensity. The ratio given for each leaf node is the intensity of the respective finding multiplied by 1,000 and without a unit because the unit may be different between different smell types. We also provide, if possible, the number of events, i.e., actions, covered by the task which is referenced by the finding. When clicking on a finding, its details are shown on the right side of the visualization. This consists of the basic smell description on the top (see system messages in Table 3) as well as the related task tree and the scene graph excerpt on the bottom. In the center of the right side are interaction elements for the manual assessment of the findings. Through them, we marked findings, e.g., as true positives and we assigned labels to them for further analysis, e.g., if they belonged to grabbing the cup in the coffee scene. We used these labels subsequently to create the statistics shown in Table 3.

The inspection was done based on the following criterion. For each type of smell, we checked, if the finding is correct with respect to what the smell is intended to find. For example, we checked if the findings for US3 – High Interaction Distance referred to tasks where the users had to overcome

[1]In the figure, US1 – Important Task is called "often executed."

a certain distance between two subsequent actions. If this was the case, we marked the finding as true positive. In this assessment, we did not consider the intensities of the findings. For example, we also marked findings for US3 – High Interaction Distance as true positive, even if the intensities showed that the distance between two subsequently used interactive objects is only a few centimeters. We did this to prevent introducing magic numbers as thresholds for deciding when a certain intensity is high enough so that the finding can be considered valid. Furthermore, focusing on only one hard criterion should reduce a potential evaluator effect that may have a higher influence in case of more and weaker assessment criteria.

Results of the Manual Inspection. Based on our inspection, we considered almost all findings as true positives. Only eight findings for US3 – High Interaction Distance in the laser mode (four for the coffee scene and four for the copier scene) were not correct. Here, the recorded coordinates for the cup and the paper for the grab and the respective ungrab event seemed to be invalid. They were far outside the local area of the scene in which the users were. This was a recording issue that resulted in physically impossible distances between two subsequent actions (several hundred virtual meters between the grab and the ungrab). As a result, the intensities of these findings were incorrect. Hence, we considered these findings as false positives.

During our manual inspection, we found a further issue that occurred only for the hand mode. In all other modes, the use and unuse events for buttons were clearly ordered. This means, if two buttons were used subsequently, then before the use event of the second button there was an unuse event for the first button. For the hand mode, this was not always the case. This is because in the hand mode, users can touch multiple buttons with one hand at the same time. This is not possible in the other modes as users cannot point at multiple buttons at the same time using a single crosshair (gaze mode) or laser pointer (laser and controller mode). Despite these differences, the findings for the usability smells were true positives.

Relating Table 3 with Table 2, we found two relevant correlations. We checked them with the Spearman's rank correlation coefficient [99]. With an increasing number of generated sequences, also the number of detected smells increases (coefficients ranging from 0.688 for US2 – Required Inefficient Actions and 0.99 for the US1 – Important Tasks). For US5 – Missing User Guidance, the smell intensities and the number of detected sequences correlate (0.924 for the prominent intensity and 0.961 for the normal intensity).

Considering the findings for US1 – Important Tasks and their distribution to the different activities in the scenes, the number of findings for the wrong button usage is higher for the hand mode in both VR scenes (42 and 72) in comparison to the other modes (at most six findings). In addition, in the hand mode the number of findings for using the wrong button is higher than for using the right button where in the other modes, it is the other way around. Hence, pressing a wrong button was done more often in the hand mode than in the other modes. The reason for this is that in the hand mode a button gets already pressed if the users accidentally wave through it with a hand, what is not possible in the other modes. For the other modes, the findings are more focused on grabbing and moving the cup or paper (more than 50% of the findings for a scene/mode combination). This shows that in these modes, the users focused on grabbing the cup and the paper.

For US2 – Required Inefficient Actions, the majority of the findings is related to grabbing and moving the cup and the paper. This is because in both scenes, the grabbed object, i.e., the cup or the paper, had to be moved via some distance in the first step and for this, the users also had to turn their heads. We observed similar results for the findings of US3 – High Interaction Distance. Here, up to all findings for a scene/mode combination are related to grabbing the cup or the paper. This reflects the scene setup as the users have to move the cup and the paper in both scenes. An additional observation for this smell is that the number of findings for using the wrong button is higher for the hand mode (13 and 57 findings) than for the other modes (at most 1 finding). Also

Table 4. Manually Captured User Issues During Our Case Study Related to the Four Interaction Modes

	Gaze	Laser	Controller	Hand
Difficulties to grab	0	0	0	6
Unintended grabbing	2	0	0	0
Unintended ungrabbing	5	4	0	12
Release in snap drop zone unclear	5	2	4	4
Unintended pressing of a button	0	0	0	16
Reset of scene required	0	0	1	12
Too far away for interaction	12	4	3	0
Required detailed help for mode usage	0	9	18	2
Users requested for improved hardware	0	0	0	4
Users considered the mode not intuitive	2	1	0	0

here, this is caused by accidental button usages and due to the fact that the buttons all have a certain distance to each other.

For US4 – Task Retried, there is no tendency in the number of findings toward a certain activity. But for this smell, our manual inspection showed that the highest intensities of findings were calculated in both scenes for the hand mode. We made this difference measurable in that we calculated the average intensity of the five findings with the highest intensities for each scene/mode combination. This average was between 2 and 18 for the other modes in comparison to 28 (coffee machine) and 29 (copier) for the hand mode. Also, the two highest intensities calculated for this smell are assigned to findings in the hand mode (76 for grabbing the cup in the coffee machine scene, 70 for grabbing the paper in the copier scene). This means, in the hand mode, there were more repeated task executions, especially for grabbing objects, and these tasks covered more recorded actions than in the other modes.

5.5 Counter Evaluation

In this article so far, we considered true positives and false positives of our findings with respect to the technical functionality of our usability smell detection. In the case study, we performed an additional analysis in which we also checked, if the true positives correctly represent usability issues and if there are any false negatives. This analysis, which we call *counter evaluation*, was based on traditional user-oriented usability engineering methods. For a practical application of our approach, it is not required to perform a counter evaluation. We did it only for evaluating the results of our approach. We describe the counter evaluation in the following.

The basic setup of our case study is already similar to a user-oriented usability evaluation as we asked our participants to perform a certain task. In addition to this, we observed our participants while using the VR scenes and took notes of the issues they had. We also recorded videos of what the users saw and did in the scenes using the screen recording tool FFSplit [27]. Based on this data, we performed the counter evaluation for identifying the actual usability issues of the VR scenes and the interaction modes. For this, we first extracted and grouped usability issues from the notes we had for each participant. We performed this extraction and grouping twice and by two separate persons to reduce an evaluator effect. Then, we counted the number of participants that had the same issues. Finally, we dropped all issues that occurred for only one participant to filter out noise. The result of this process were 10 usability issues. Those are listed in the left column of Table 4. The right four columns of the table show how often a specific issue came up for the different interaction modes.

Table 5. Time Spent in Seconds (Standard Deviations in Parentheses) by Participants for Individual Parts of the Respective Tasks in the VR Scenes

Scene	Action	Gaze		Laser		Controller		Hand	
		μ	σ	μ	σ	μ	σ	μ	σ
Coffee scene	Move cup	13	(10)	15	(13)	7	(6)	19	(28)
	Press button	22	(14)	21	(13)	34	(16)	20	(17)
Copier scene	Move paper	25	(17)	38	(33)	39	(22)	57	(29)
	Press button	26	(15)	21	(12)	30	(18)	13	(9)

Based on the table, we made the following observations. Considering grabbing and ungrabbing (first three issues), 18 issues occurred for the hand mode followed by 7 for the gaze, and 4 for the laser mode. For the controller mode, no grabbing issues were observed. Via all modes, 15 participants had problems in understanding the so called *snap drop zones*. These are the intended destination positions of the cup and the paper in the scenes. When a user releases an object close to these positions (automatic release in gaze mode), the object places itself in these positions, which may also mean turning itself back to a realistic orientation. But some participants tried to correct the position as to them it did not seem 100% correct.

16 participants pressed at least one button unintentionally. This usually happened when they tried to interact with one button, but unintentionally strived another button at the same time. This was only possible in the hand mode as already described in the previous section. For our case study, this was not really an issue as only one button in the copier scene and three buttons in the coffee scene had an assigned functionality and this functionality in the coffee scene was also the same for all three buttons. But if the unintentionally pressed buttons had other functions, then this was a rather severe usability issue considering this mode. In addition, we saw throughout all interaction modes that many participants initially tried wrong buttons for interacting with the device (28 in the copier scene and 38 in the coffee scene). These issues are not related to a certain interaction mode as they are caused by a user's misunderstanding of the respective device. Hence, we did not list them in the above table.

12 participants for the hand mode and 1 participant for the controller mode required a reset of the scene. For the other modes, this was not required, although 1 participant accidentally triggered the scene reset multiple times in the laser mode because of pressing all buttons on the controller when interacting, including the grip button. This participant was also the only participant who did not finalize the task execution.

19 participants were too far away from the interactive objects when they started the interaction and, therefore, we asked them to step a bit closer. This was more often an issue for the gaze mode (12 participants) than for the laser and controller mode (4 and 3 participants). For the hand mode, this was no issue at all because an interaction in this mode is only possible by stepping close to the interactive objects. 29 participants required detailed help for how to use the respective mode. Most help was required for the controller mode (18 participants) and no help was required for the gaze mode. Finally, 4 participant stated that the hand mode requires technical improvement so that it is easier to use, and 3 participants considered the gaze and laser mode as not intuitive.

In addition to the participants' issues, we measured the average time the users spent on different task parts in the two VR scenes. For this, we used the screen recordings as input. The corresponding periods are listed in seconds (standard deviations in parentheses) in Table 5. The rows of the table show the VR scene and the respective task part. The columns represent the different interaction modes. For moving the cup in the coffee scene as well as moving the paper in the copier scene, the participants took longest with the hand mode (19 and 57 seconds). In this respect, the controller

mode worked best for the coffee machine scene (7 seconds) and the gaze mode worked best for the copier scene (25 seconds). A pairwise two-sided Wilcoxon test [98] on the measurements with a Hochberg p-value adjustment [45] for both scenes showed that only the difference between the gaze and the hand mode in the copier scene is significant (p-value of 0.0016). In contrast, for pushing buttons on the devices in both scenes, the hand mode was fastest (20 and 12 seconds), whereas the controller mode worked worst (34 and 30 seconds). The latter fact seems to be in line with the number of participants that required detailed help for this mode in comparison to the other modes. Here, a pairwise Wilcoxon test with a Hochberg p-value adjustment showed that only the differences between the hand and the gaze mode, as well as between the hand and the controller mode in the copier scene are significant considering a minimum p-value of 0.005 [11].

5.6 Discussion

Based on the results of our case study, we can answer our research questions. RQ 1 asks, how and to what extent must the usage of a VR be recorded and stored for a subsequent task tree generation. Our case study showed that we were able to generate task trees based on the data we recorded. Hence, our recording mechanism has shown to provide the input data as required for the task tree generation. Therefore, we answer the question with the statement that for this purpose, it is required to record at least the grabbing and using of virtual objects as well as head movements. For analyses others than ours, other actions may need to be recorded, but this cannot be stated for sure based on our case study.

We also saw in our results similar relationships between the recorded actions and the generated task trees as we saw in our previous work [37, 39–41]. For example, the number of task trees increases with an increasing number of recorded actions. In addition, the subset of prominent sequences already represents a rather large part of the recorded data, which was also the case in our previous work. This allows us to answer RQ 2, which asks how and to what extent can task trees be generated from recordings of VR usage? We can generate task trees using the approach described in [37], [40], and [41] and the characteristics of these task trees, e.g., their representativeness for the recorded user behavior, are similar as for websites and desktop software.

The most important research question for this work is RQ 3. The first aspect of this question asks how can usability issues in a VR reliably be derived from generated task trees? For answering the question, we adapted the usability smell detection from our previous work [37, 38] for the use with VR. We considered almost all findings of our approach as true positives considering the technical point of view. For the second aspects of RQ 3, i.e., the extent to which usability issues can be detected by our approach, we compared our findings with the results of the traditional usability evaluation, i.e., our counter evaluation. Here, we see diverse overlaps between the issues found by both approaches. Those are listed in Table 6. In the left column, the table shows groups of usability issues resulting from our counter evaluation. The center column refers to the interaction mode or user activity where this issue was most severe. The right column lists the effects of these issues as they can be seen in our usability smell findings.

Based on these overlaps between the findings of our proposed usability smell detection and the usability issues stemming from the counter evaluation, we can answer the second aspect of RQ 3 as follows. First of all, we derive that if a usability issue exists, then also the number of findings of our usability smell detection increases as well as the intensities of US5 – Missing User Guidance. Hence, both aspects are indicators for usability issues. But we can be even more precise for the individual smells.

Due to their number and their relation to the different activities in the VR scenes, we conclude that the findings for US1 – Important Tasks can be helpful to identify unexpected activities like the pressing of the wrong buttons on the devices. An inspection of referenced tasks allows an

Table 6. Overlaps between the Usability Issues Found by the Counter Evalution and the Findings of Our Usability Smell Detection

Usability issues	Most problematic mode/activity	Effect in smell findings
Issues related to grabbing found through the counter evaluation —Difficulties to grab —Unintended grabbing —Unintended ungrabbing —Release in snap drop zone unclear —Reset of scene required —Long time spent for grabbing	Hand mode	—Increased number of findings and highest intensities of the findings for US4 - Task Retried in the hand mode —Highest intensities for the two findings of US5 - Missing User Guidance for the hand mode
Issues related to pressing buttons found through the counter evaluation —Pressing of wrong buttons —Unintended pressing of buttons	Hand mode (for unintended pressing of buttons)	—Increased number of findings for US1 - Important Tasks, US3 - High Interaction Distance, and US4 - Task Retried for button usage, especially for wrong buttons, in the hand mode
Issues related to high interaction distance stemming from the case study setup —Moving objects over a certain distance	Moving cup/paper	—Increased number of findings for US2 - Required Inefficient Actions and US3 - High Interaction Distance for the respective activities —Higher intensities of findings for US3 - High Interaction Distance for the respective activities
General issues found through the counter evaluation —Required detailed help for mode usage —Too far away for interaction —Users requested for improved hardware —Users considered the mode not intuitive	All modes	—Not visible in the findings of the usability smell detection

evaluator to find out in which situations these activities were performed. Not directly visible is the cause for the execution of these activities, which could be accidentally or intended but wrong due to misconceptions. In our case, they were triggered accidentally when users waved their hands through the button sets on the virtual devices. But in other situations, the unexpected activities may be intended by users. Then, an evaluator can draw other conclusions from the findings for this smell, e.g., that users think a certain activity is required.

The findings of US2 – Required Inefficient Actions correctly identified the requirement for moving the head when moving the cup or the paper. In addition, there are less findings for the other user activities where less head movements are required. Most importantly, the findings with the highest intensities showed up for tasks, which should be executed with a high efficiency. Hence, this smell is able to provide helpful findings. We did not observe findings with high intensities for tasks of exploring the VR. The reason for this may be that our VR scenes did not provide much content to explore. So we cannot judge if the smell would be as helpful for VRs with more intended exploration. In addition, our manual inspection showed a requirement for revising the logging of head movements. For example, we had a finding for a task consisting of three leaf nodes, two for using and unusing a button and a third for head movements, but this task already had an intensity which indicated that 70.4% of the actions executed for this task were head movements. This can only be caused by a high number of repeated head movement events based on which this task was generated.

In our VR scenes, the users first had to move the cup or the paper via a certain distance. The focus of the findings for US3 – High Interaction Distance on the same activities shows that this smell is able to detect usability issues related to distances. Problematic are situations in which the coordinates of the grabbed objects are wrong as happened for the laser mode. This requires an improvement of the logging but is not an issue for the smell detection itself.

The manually inspected intensities for the findings of US4 – Task Retried are highest for the hand mode. This is in line with the issues the users had when grabbing the cup and the paper in the hand mode. From this, we conclude that this smell provides helpful results for finding issues where users need to retry a certain activity because it is hard to perform.

Our analyses also showed that depending on the concrete type of smell, other information may be of importance. For example, for one smell, the number of findings can be an indicator for usability issues, whereas for other smells, the intensities are of most relevance. Definitely required is a possibility to separate more relevant findings from less relevant because with an increasing number of recorded events, also the number of detected task trees and found usability smells increases. This may make a handling of a list of findings rather cumbersome. Such a separation can be done, e.g., by considering only those findings referring to prominent sequences or by defining thresholds for the smell intensities. In addition, it would be helpful to filter duplicate findings.

Our case study was set up so that different interaction modes were used. This shows, that our approach can be used for different interaction modes. In addition, the findings match the issues of the users which were either mode or scene specific. This shows that our approach is able to handle at least the four implemented interaction modes and that it is capable of identifying issues unrelated to a selected interaction mode.

As shown by the last row of Table 6, our approach is not able to detect rather general issues, e.g., if users have a misunderstanding or if they consider something as not intuitive. This was also already an issue in our previous work, where we applied a similar approach on websites. Hence, our work can only contribute to a usability evaluation, but it cannot be the only method to use. In addition, we utilize recordings of software usage what can also be done without awareness of the user. Hence, when applying our approach, ethical issues and a possibility for opting-out by the user should be considered.

5.7 Threats to Validity

The evaluation of our approach underlies certain threats to validity. First of all, it depends on intensive software implementation which may contain bugs even after in-depth testing. But considering that our results are reasonable, we assume that remaining bugs except the one for recording object coordinates in the laser mode do not have a strong effect.

Furthermore, our two VR scenes are rather small with only a limited set of user activities and freedom to interact. They do not cover different locomotion techniques, further object interactions, text input, or tasks including an exploration of a VR. This allowed for a good comparability of the results of our approach for different interaction modes in VRs. We cannot estimate how our results will change in case of larger applications, more locomotion and exploration, as well as further object interaction techniques or text input. In addition, although we do not expect this because of our experience with the task tree generation stemming from our previous research, larger VR case studies may lead to a data explosion and corresponding processing issues.

Most importantly, we compared the results of our approach with the results of the application of traditional user testing. Because both approaches were applied on the same set of users, we can consider that our approach provides valid results as they correlate with the results of the user testing. But it can also be the case, that our specific case study setup itself caused this correlation and that this correlation may not exist for other case studies.

6 CONCLUSION AND OUTLOOK

We described an approach for automated usability evaluation of VR applications. The approach is based on recordings of user actions, a subsequent generation of task trees, and the detection of five usability smells. With a rather large case study, we showed that the approach is able to detect usability issues and to assess diverse aspects of a VR's usability. For example, we were able to identify problems when moving objects as well as inefficiencies in user tasks. Due to its full automation, our approach may reduce the effort for performing VR usability evaluations. Despite these aspects, we also uncovered limitations, e.g., that certain misunderstandings of users cannot be detected. Therefore, the approach can contribute to usability engineering of VR applications and serve as a regularly applied tooling, but should be combined with other methods.

In future work, the approach could be extended with further usability smells. For example, a smell could check for the time spent for a certain detected task and if this time is larger than it would be expected for the tasks structure. Furthermore, the approach should be applied in larger case studies with more complex VRs and including more locomotion. In addition, our work could be extended to include further actions and events. For example, it would be interesting to record the usage of controller buttons and to separate grab and use actions from head movements. This would result in multiple separate event streams. An adapted algorithm would then allow to generate task trees representing parallel user actions. Based on this, new usability smells focusing on parallelism could be considered. In addition, parallel user actions that need to be considered as efficient could be treated as such. Finally, we intend to perform a study with users of our approach to check, if they are able to correctly understand the usability issues and if they would be able to solve them even without being experts in our approach or in usability engineering.

7 REPLICATION KIT

All the data we recorded in the case study, the performed statistical tests, as well as the software for detecting the usability smells have been published in a replication kit available at https://doi.org/10.5281/zenodo.894173.

ACKNOWLEDGMENTS

We thank all the volunteers that participated in our case study, either as participant or as supporter, as well as all our proof readers for their valuable feedback.

REFERENCES

[1] 1998. ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability (ISO 9241-11:1998).

[2] 2018. VRTK - Virtual Reality Toolkit. Retrieved 1, 2018 from https://vrtoolkit.readme.io/.

[3] Diogo Almeida, José Creissac Campos, João Saraiva, and João Carlos Silva. 2015. Towards a catalog of usability smells. In *Proceedings of the Volume I: Artificial Intelligence and Agents, Distributed Systems, and Information Systems (ACM SAC'15)*. ACM, 175–181.

[4] Saleema Amershi, Jalal Mahmud, Jeffrey Nichols, Tessa Lau, and German Attanasio Ruiz. 2013. LiveAction: Automating web task model generation. *ACM Trans. Interact. Intell. Syst.* 3, 3, Article 14 (Oct. 2013), 23 pages. DOI: https://doi.org/10.1145/2533670.2533672

[5] Richard Atterer. 2008. *Usability Tool Support for Model-Based Web Development.* dissertation. http://nbn-resolving.de/urn:nbn:de:bvb:19-92963.

[6] Fiora T. W. Au, Simon Baker, Ian Warren, and Gillian Dobbie. 2008. Automated usability testing framework. In *Proceedings of the 9th Conference on Australasian User Interface - Volume 76 (AUIC'08)*. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 55–64. http://dl.acm.org/citation.cfm?id=1378337.1378349.

[7] Rebecca Baker. 2014. Remote Usability Testing: Thinking Outside the Lab. Retrieved 11, 2017 from http://www.uxmatters.com/mt/archives/2014/04/remote-usability-testing-thinking-outside-the-lab.php.

[8] S. Baker, F. Au, G. Dobbie, and I. Warren. 2008. Automated usability testing using HUI analyzer. In *Proceedings of the 19th Australian Conference on Software Engineering (ASWEC'08)*. 579–588. DOI: https://doi.org/10.1109/ASWEC.2008.4483248

[9] Aurora Bedford. 2015. Don't Prioritize Efficiency Over Expectations. Retrieved 9, 2017 from http://www.nngroup.com/articles/efficiency-vs-expectations/.

[10] Aurora Bedford. 2015. No More Pogo Sticking: Protect Users from Wasted Clicks. Retrieved 9, 2017 from http://www.nngroup.com/articles/pogo-sticking/.

[11] Daniel J. Benjamin, James O. Berger, Magnus Johannesson, Brian A. Nosek, Eric-Jan Wagenmakers, Richard Berk, Kenneth A. Bollen, Björn Brembs, Lawrence Brown, Colin Camerer, et al. 2017. Redefine statistical significance. *Nat. Hum. Behav.* (2017).

[12] Nicholas Benson. 2017. Interaction Engine. Retrieved 9, 2017 from https://github.com/leapmotion/UnityModules/wiki/Interaction-Engine.

[13] Bootstrap core team and contributors. 2017. Bootstrap. Retrieved 11, 2017 from https://getbootstrap.com/.

[14] Doug A. Bowman, Joseph L. Gabbard, and Deborah Hix. 2002. A survey of usability evaluation in virtual environments: Classification and comparison of methods. *Presence: Teleoper. Virtual Environ.* 11, 4 (Aug. 2002), 404–424. DOI: https://doi.org/10.1162/105474602760204309

[15] Paolo Burzacca and Fabio Paternò. 2013. Remote usability evaluation of mobile web applications. In *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments*, Masaaki Kurosu (Ed.). Lecture Notes in Computer Science, Vol. 8004. Springer Berlin Heidelberg, 241–248. DOI: https://doi.org/10.1007/978-3-642-39232-0_27

[16] Tonio Carta, Fabio Paternò, and Vagner Santana. 2011. Support for remote usability evaluation of web mobile applications. In *Proceedings of the 29th ACM International Conference on Design of Communication (SIGDOC'11)*. ACM, New York, NY, 129–136. DOI: https://doi.org/10.1145/2038476.2038502

[17] Cátedra SAES de la Universidad de Murcia. 2014. OHT Plus: An application framework for non-intrusive usability testing tools. Retrieved 6, 2014 from http://www.catedrasaes.org/wiki/OHTPlus.

[18] Selem Charfi, Houcine Ezzedine, Christophe Kolski, and Faouzi Moussa. 2011. Towards an automatic analysis of interaction data for HCI evaluation: Application to a transport network supervision system. In *Proceedings of the 14th International Conference on Human-Computer Interaction: Design and Development Approaches - Volume Part I (HCII'11)*. Springer-Verlag, Berlin, Heidelberg, 175–184. http://dl.acm.org/citation.cfm?id=2022384.2022407.

[19] HTC Corporation. 2017. Vive. Retrieved 9, 2017 from https://www.vive.com/.

[20] Allen Cypher. 1991. EAGER: Programming repetitive tasks by example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'91)*. ACM, New York, NY, 33–39. DOI: https://doi.org/10.1145/108844.108850

[21] Vagner Figuerêdo de Santana and Maria Cecília Calani Baranauskas. 2015. WELFIT: A remote evaluation tool for identifying Web usage patterns through client-side logging. *Int. J. Hum.-Comput. Studies* 76 (2015), 40–49.

[22] Deloitte. 2016. Head Mounted Displays in deutschen Unternehmen - Ein Virtual, Augmented und Mixed Reality Check. Retrieved 11, 2017 from http://www2.deloitte.com/de/de/pages/technology-media-and-telecommunications/articles/head-mounted-displays-in-deutschen-unternehmen.html.

[23] Alexiei Dingli and Justin Mifsud. 2011. USEFul: A framework to mainstream web site usability through automated evaluation. *Int. J. Hum. Comput. Interact.* 2, 1 (2011), 10–30. http://cscjournals.org/csc/manuscript/Journals/IJHCI/volume2/Issue1/IJHCI-19.pdf.

[24] Ralf Dörner, Wolfgang Broll, Paul Grimm, and Bernhard Jung (Eds.). 2013. *Virtual und Augmented Reality (VR / AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. Springer Berlin Heidelberg, Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-28903-3

[25] Arianna D'Ulizia, Fernando Ferri, and Patrizia Grifoni. 2011. A survey of grammatical inference methods for natural language learning. *Artif. Intell. Rev.* 36, 1 (Jun. 2011), 1–27. DOI: https://doi.org/10.1007/s10462-010-9199-1

[26] Sebastian Feuerstack, Marco Blumendorf, Maximilian Kern, Michael Kruppa, Michael Quade, Mathias Runge, and Sahin Albayrak. 2008. Automated usability evaluation during model-based interactive system development. In *Proceedings of the 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams (HCSE-TAMODIA'08)*. Springer-Verlag, Berlin, Heidelberg, 134–141.

[27] FFsplit. 2017. FFSPLIT. Retrieved 9, 2017 from http://www.ffsplit.com.

[28] Software Engineering for Distributed System. 2017. AutoQUEST – Testing, Analysing, and Observing Event-driven Software. Retrieved 9, 2017 from https://autoquest.informatik.uni-goettingen.de/.

[29] Peter Géczy, Noriaki Izumi, Shotaro Akaho, and Kôiti Hasida. 2007. Usability analysis framework based on behavioral segmentation. In *E-Commerce and Web Technologies*, Giuseppe Psaila and Roland Wagner (Eds.). Lecture Notes in Computer Science, Vol. 4655. Springer Berlin Heidelberg, 35–45. DOI: https://doi.org/10.1007/978-3-540-74563-1_4

[30] Steven Gomez and David Laidlaw. 2012. Modeling task performance for a crowd of users from interaction histories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'12)*. ACM, New York, NY, 2465–2468. DOI: https://doi.org/10.1145/2207676.2208412

[31] Google. 2017. Google Analytics. Retrieved 12, 2017 from http://www.google.com/analytics/.

[32] Google. 2018. Test auf Optimierung für Mobilgeräte. Retrieved 1, 2018 from https://search.google.com/test/mobile-friendly.

[33] Google LLC. 2017. Android Developers. Retrieved 11, 2017 from https://developer.android.com/index.html.

[34] Julián Grigera, Alejandra Garrido, and José Matías Rivero. 2014. A tool for detecting bad usability smells in an automatic way. In *Web Engineering*, Sven Casteleyn, Gustavo Rossi, and Marco Winckler (Eds.). Lecture Notes in Computer Science, Vol. 8541. Springer International Publishing, 490–493. DOI: https://doi.org/10.1007/978-3-319-08245-5_34

[35] Julián Grigera, Alejandra Garrido, José Matías Rivero, and Gustavo Rossi. 2017. Automatic detection of usability smells in web applications. *Int. J. Hum.-Comput. Studies* 97 (2017), 129–148. DOI: https://doi.org/10.1016/j.ijhcs.2016.09.009

[36] Karl Grooves. 2007. The Limitations of Server Log Files for Usability Analysis. Retrieved 6, 2015 from http://boxesandarrows.com/the-limitations-of-server-log-files-for-usability-analysis/.

[37] Patrick Harms. 2016. *Automated Field Usability Evaluation Using Generated Task Trees*. Ph.D. Dissertation. http://hdl.handle.net/11858/00-1735-0000-0028-8684-1.

[38] Patrick Harms and Jens Grabowski. 2014. Usage-based automatic detection of usability smells. In *Human-Centered Software Engineering*, Stefan Sauer, Cristian Bogdan, Peter Forbrig, Regina Bernhaupt, and Marco Winckler (Eds.). Lecture Notes in Computer Science, Vol. 8742. Springer Berlin Heidelberg, 217–234. DOI: https://doi.org/10.1007/978-3-662-44811-3_13

[39] Patrick Harms and Jens Grabowski. 2015. Consistency of task trees generated from website usage traces. In *Proceedings of the 17th International Conference on System Design Languages (SDL'15)*. Springer Berlin Heidelberg, 8. DOI: https://doi.org/10.1007/978-3-319-24912-4_9

[40] Patrick Harms, Steffen Herbold, and Jens Grabowski. 2014. Extended trace-based task tree generation. *Int. J. Adv. Intel. Syst.* 7, 3, 4 (Dec. 2014), 450–467. http://www.iariajournals.org/intelligent_systems/.

[41] Patrick Harms, Steffen Herbold, and Jens Grabowski. 2014. Trace-based task tree generation. In *Proceedings of the 7th International Conference on Advances in Computer-Human Interactions (ACHI'14)*. XPS - Xpert Publishing Services, 6.

[42] N. Harrati, I. Bouchrika, A. Tari, and A. Ladjailia. 2015. Automating the evaluation of usability remotely for web applications via a model-based approach. In *Proceedings of the 1st International Conference on New Technologies of Information and Communication (NTIC'15)*. 1–6. DOI: https://doi.org/10.1109/NTIC.2015.7368757

[43] Marcus Hegner. 2003. *Methoden zur Evaluation von Software*. IZ, InformationsZentrum Sozialwiss. http://books.google.de/books?id=NhnMHAAACAAJ.

[44] David M. Hilbert and David F. Redmiles. 2000. Extracting usability information from user interface events. *ACM Comput. Surv.* 32, 4 (Dec. 2000), 384–421. DOI: https://doi.org/10.1145/371578.371593

[45] Yosef Hochberg. 1988. A sharper bonferroni procedure for multiple tests of significance. 75 (Dec. 1988), 800–802.

[46] Scott E. Hudson, Bonnie E. John, Keith Knudsen, and Michael D. Byrne. 1999. A tool for creating predictive performance models from user interface demonstrations. In *Proceedings of the 12th Annual ACM Symposium on User Interface Software and Technology (UIST'99)*. ACM, New York, NY, 93–102.

[47] Melody Y. Ivory and Marti A. Hearst. 2001. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.* 33, 4 (Dec. 2001), 470–516. DOI: https://doi.org/10.1145/503112.503114

[48] Anthony Jameson, Angela Mahr, Michael Kruppa, Andreas Rieger, and Robert Schleicher. 2007. Looking for unexpected consequences of interface design decisions: The MeMo workbench. In *Task Models and Diagrams for User Interface Design*, Marco Winckler, Hilary Johnson, and Philippe Palanque (Eds.). Lecture Notes in Computer Science, Vol. 4849. Springer Berlin, Heidelberg, 279–286.

[49] Kasper Løvborg Jensen and Lars Bo Larsen. 2007. Evaluating the usefulness of mobile services based on captured usage data from longitudinal field trials. In *Proceedings of the 4th International Conference on Mobile Technology, Applications, and Systems and the 1st International Symposium on Computer Human Interaction in Mobile Technology (Mobility'07)*. ACM, New York, NY, 675–682. DOI: https://doi.org/10.1145/1378063.1378177

[50] Bonnie E. John. 2015. CogTool. Retrieved 6, 2015 from http://cogtool.com/.

[51] Bonnie E. John, Konstantine Prevas, Dario D. Salvucci, and Ken Koedinger. 2004. Predictive human performance modeling made easy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'04)*. ACM, New York, NY, 455–462.

[52] Christos Katsanos, Nikolaos Tselios, and Nikolaos Avouris. 2006. InfoScent evaluator: A semi-automated tool to evaluate semantic appropriateness of hyperlinks in a web site. In *Proceedings of the 18th Australia Conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments (OZCHI'06)*. ACM, New York, NY, 373–376. DOI: https://doi.org/10.1145/1228175.1228249

[53] Jun H. Kim, Daniel V. Gunn, Eric Schuh, Bruce Phillips, Randy J. Pagulayan, and Dennis Wixon. 2008. Tracking real-time user experience (TRUE): A comprehensive instrumentation solution for complex systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. ACM, New York, NY, 443–452. DOI: https://doi.org/10.1145/1357054.1357126

[54] Ralph Krimmel. 2014. Improving Automatic Task Tree Generation With Alignment Algorithms.

[55] Inc. Leap Motion. 2017. Unity. Retrieved 9, 2017 from https://developer.leapmotion.com/unity/.

[56] Andreas Lecerof and Fabio Paternò. 1998. Automatic support for usability evaluation. *IEEE Trans. Softw. Eng.* 24, 10 (Oct. 1998), 863–888.

[57] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan. 2004. UsiXML: A user interface description language for context-sensitive user interfaces. In *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages,"* 55–62.

[58] Matomo.org. 2018. Matomo – Liberating Analytics. Retrieved 1, 2018 from https://matomo.org/.

[59] David Maulsby. 1997. Inductive task modeling for user interface customization. In *Proceedings of the 2nd International Conference on Intelligent User Interfaces (IUI'97)*. ACM, New York, NY, 233–236. DOI: https://doi.org/10.1145/238218.238331

[60] Thomas Memmel. 2009. *User Interface Specification for Interactive Software Systems - Process-, Method- and Tool-Support for Interdisciplinary and Collaborative Requirements Modelling and Prototyping-Driven User Interface Specification*. PhD thesis. University of Konstanz.

[61] Microsoft. 2018. Das neue Windows Mixed Reality. Retrieved 1, 2018 from https://www.microsoft.com/de-de/windows/windows-mixed-reality.

[62] C. Moser. 2012. *User Experience Design: Mit erlebniszentrierter Softwareentwicklung zu Produkten, die begeistern*. Springer Berlin Heidelberg.

[63] Donald A. Norman. 2002. *The Design of Everyday Things* (1. basic paperback ed., [repr..] ed.). Basic Books, [New York, NY].

[64] Oculus. 2018. oculus rift. Retrieved 1, 2018 from https://www.oculus.com/rift/.

[65] U.S. Dept. of Health and Human Services. 2006. The Research-Based Web Design & Usability Guidelines, Enlarged/Expanded edition. Retrieved 7, 2017 from http://guidelines.usability.gov/.

[66] Gary M. Olson, James D. Herbsleb, and Henry H. Rueter. 1994. Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Hum-Comput. Interact.* 9 (1994), 427–472.

[67] Oracle. 2017. Trail: Creating a GUI With JFC/Swing. Retrieved 11, 2017 from https://docs.oracle.com/javase/tutorial/uiswing/.

[68] Fabio Paternò. 2003. ConcurTaskTrees: An engineered notation for task models. In *The Handbook of Task Analysis for Human Computer Interaction*, Dan Diaper and Neville Stanton (Eds.). Lawrence Erlbaum Associates Publishers, 483–503.

[69] Fabio Paternò. 2003. Tools for remote web usability evaluation. In *Proceedings of the 10th International Conference on Human-Computer Interaction.*, Vol. 1. Erlbaum, 828–832. http://giove.isti.cnr.it/attachments/publications/2003-A2-95.pdf.

[70] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. 1997. ConcurTaskTrees: A diagrammatic notation for specifying task models. In *Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction (INTERACT'97)*. Chapman & Hall, Ltd., London, UK, 362–369. DOI: https://doi.org/10.1007/978-0-387-35175-9_58

[71] Fabio Paternò, Angela Piruzza, and Carmen Santoro. 2005. Remote usability analysis of MultiModal information regarding user behaviour. InInternational COST 294 Workshop on User Interface Quality Models, pp. 15–22. http://giove.isti.cnr.it/attachments/publications/2005-A2-134.pdf.

[72] Fabio Paternò, Angela Piruzza, and Carmen Santoro. 2007. Remote web usability evaluation exploiting multimodal information on user behavior. In *Computer-Aided Design of User Interfaces V*, Gaëlle Calvary, Costin Pribeanu, Giuseppe Santucci, and Jean Vanderdonckt (Eds.). Springer Netherlands, 287–298. DOI: https://doi.org/10.1007/978-1-4020-5820-2_24

[73] Fabio Paternò, Andrea Russino, and Carmen Santoro. 2007. Remote evaluation of mobile applications. In *Task Models and Diagrams for User Interface Design*, Marco Winckler, Hilary Johnson, and Philippe Palanque (Eds.). Lecture Notes in Computer Science, Vol. 4849. Springer, Berlin, 155–169. DOI: https://doi.org/10.1007/978-3-540-77222-4_13

[74] Fabio Paternò, Antonio Giovanni Schiavone, and Antonio Conti. 2017. Customizable automatic detection of bad usability smells in mobile accessed web applications. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services (MobileHCI'17)*. ACM, New York, NY, Article 42, 11 pages. DOI: https://doi.org/10.1145/3098279.3098558

[75] Michael Quade, Marco Blumendorf, and Sahin Albayrak. 2010. 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*.

[76] Neil Ramsay, Stuart Marshall, and Alex Potanin. 2008. Annotating UI architecture with actual use. In *Proceedings of the Ninth Conference on Australasian User Interface – Volume 76 (AUIC'08)*. Australian Computer Society, Inc., Darlinghurst, Australia, 75–78. http://dl.acm.org/citation.cfm?id=1378337.1378351.

[77] Karen Renaud and Phil Gray. 2004. Making sense of low-level usage data to understand user activities. In *Proceedings of the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries (SAICSIT'04)*. South African Institute for Computer Scientists and Information Technologists, Republic of South Africa, 115–124. http://dl.acm.org/citation.cfm?id=1035053.1035067.

[78] Michael Richter and Markus D. Flückiger. 2013. *Usability Engineering kompakt: Benutzbare Software gezielt entwickeln*. Springer Berlin, Heidelberg.

[79] Joy Robinson, Candice Lanius, and Ryan Weber. 2017. The past, present, and future of UX empirical research. *Commun. Des. Q. Rev* 5, 3 (Nov. 2017), 10–23.

[80] Stephanie Rosenbaum. 2008. The future of usability evaluation: Increasing impact on value. In *Maturing Usability*, Effie Lai-Chong Law, Ebba Thora Hvannberg, and Gilbert Cockton (Eds.). Springer, 344–378. http://dblp.uni-trier.de/db/series/hci/LawHC08.html#Rosenbaum08.

[81] Jean-David Ruvini et al. 2000. APE: Learning user's habits to automate repetitive tasks. In *Proceedings of the Conference on Intelligent User Interfaces (IUI'00)*. 229–232.

[82] Dario D. Salvucci. 2009. Rapid prototyping and evaluation of in-vehicle interfaces. *ACM Trans. Comput.-Hum. Interact.* 16, 2, Article 9 (June 2009), 33 pages. DOI: https://doi.org/10.1145/1534903.1534906

[83] Florian Sarodnick and Henning Brau. 2006. *Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung* (1st ed.). Huber, Bern.

[84] Andrew Sears. 1995. AIDE: A step toward metric-based interface development tools. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST'95)*. ACM, New York, NY, 101–110. DOI: https://doi.org/10.1145/215585.215704

[85] Antonio C. Siochi and Roger W. Ehrich. 1991. Computer analysis of user interfaces based on repetition in transcripts of user sessions. *ACM Trans. Inf. Syst.* 9, 4 (Oct. 1991), 309–335. DOI: https://doi.org/10.1145/119311.119312

[86] Antonio C. Siochi and Deborah Hix. 1991. A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'91)*. ACM, New York, NY, 301–305. DOI: https://doi.org/10.1145/108844.108926

[87] Jenifer Tidwell. 2010. *Designing Interfaces – Patterns for Effective Interaction Design* (2nd ed.). O'Reilly Media, Incorporated. http://books.google.de/books?id=5gvOU9X0fu0C.

[88] Thomas Tiedtke, Christian Märtin, and Norbert Gerth. 2002. AWUSA – A Tool for Automated Website Usability Analysis. In *Proceedings of the 9th International Workshop on the Design, Specification and Verification of Interactive Systems (DSV-IS'02)*.

[89] H. Trætteberg. 2002. *Model-based User Interface Design*. Information Systems Group, Department of Computer and Information Sciences, Faculty of Information Technology, Mathematics and Electrical Engineering, Norwegian University of Science and Technology.

[90] Thomas Tullis and William Albert. 2008. *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*. Morgan Kaufmann Publishers Inc., San Francisco, CA.
[91] Unity Technologies. 2017. Unity 3D. Retrieved 9, 2017 from https://unity3d.com.
[92] Martijn Van Welie, Gerrit C. Van Der Veer, and Anton Eliëns. 1998. An ontology for task world models. In *Proceedings of Design, Specification and Verification of Interactive Systems (DSV-IS'98), Abingdon*. http://citeseerx.ist.psu.edu/viewdoc/summary.
[93] Google VR. 2017. Google Cardboard. Retrieved 9, 2017 from https://vr.google.com/cardboard/.
[94] Google VR. 2017. Google Daydream. Retrieved 9, 2017 from https://vr.google.com/daydream/.
[95] VRgluv. 2018. VRgluv. Retrieved 1, 2018 from https://vrgluv.com/.
[96] W3Schools. 2017. HTML5 Tutorial. Retrieved 11, 2017 from https://www.w3schools.com/html/default.asp.
[97] G. Wallner and S. Kriglstein. 2014. PLATO: A visual analytics system for gameplay data. *Comput. Graph.* 38 (2014), 341–356. DOI: https://doi.org/10.1016/j.cag.2013.11.010
[98] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biomet. Bull.* 1, 6 (1945), 80–83. http://www.jstor.org/stable/3001968.
[99] D. Zwillinger and S. Kokoska. 1999. *CRC Standard Probability and Statistics Tables and Formulae*. CRC.

Received January 2018; revised October 2018; accepted December 2018