Discovery of Interaction Patterns with Graphical User Interface Usage Mining

Entdeckung von Interaktionsmustern bei Verwendung der grafischen Benutzeroberfläche

Master-Thesis von Markus Schröder aus Bensheim
Tag der Einreichung:

1. Gutachten: Prof. Dr. Max Mühlhäuser
2. Gutachten: Dr. Benedikt Schmidt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Telecooperation Group

Discovery of Interaction Patterns with Graphical User Interface Usage Mining
Entdeckung von Interaktionsmustern bei Verwendung der grafischen Benutzeroberfläche

Vorgelegte Master-Thesis von Markus Schröder aus Bensheim

1. Gutachten: Prof. Dr. Max Mühlhäuser
2. Gutachten: Dr. Benedikt Schmidt

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den January 20, 2015

(Markus Schröder)

Abstract

The Graphical User Interface (GUI) serves as an interface to directly manipulate graphical elements. Application softwares contain these graphical elements which represent functionality for a specific application. The main target is the discovery of patterns which express functionality usage. In the first part, the interaction between study participants and application softwares are observed by a tool called "Interaction Observer". This is accomplished with the Accessibility technology which provides access to graphical elements. The approach is termed "Graphical Software Mining" which mines software exclusively on a graphical level. The observation forms an interaction log. In the second part, interaction patterns are defined and in the interaction log discovered. Interaction patterns are sequences of interactions which express an intentional action. This approach is termed "GUI Usage Mining" which discovers usage information from GUI interaction logs. Initially, reference patterns are discovered by human annotators. The insights are used to preprocess the raw interaction log. Transactions, which are meaningful interaction clusters, are identified with four approaches. This enables the application of four mining strategies: sequential pattern mining, graph mining, process mining and mining based on n-grams. They are implemented with the help of extern libraries. User-program pairs are analyzed in the evaluation. The patterns of the four strategies are judged by the corresponding participants. The evaluation indicates that the n-gram based strategy discovers more accepted patterns.

Zusammenfassung

Die grafische Benutzeroberfläche dient als Schnittstelle, um grafische Elemente direkt zu manipulieren. Programme enthalten diese grafischen Elemente, die Funktionalität für eine bestimmte Anwendung repräsentieren. Das hauptsächliche Ziel ist die Entdeckung von Interaktionsmustern, die die Benutzung der Funktionalität ausdrücken. Im ersten Teil wird die Interaktion zwischen Studienteilnehmern und Anwendungen mit einem Werkzeug, dem sog. Interaktionsbeobachter, beobachtet. Dies wird über Barrierefreiheit-Technologie, die Zugang zu grafischen Elementen bietet, bewerkstelligt. Dieses Vorgehen wird *Graphical Software Mining* genannt, das ausschließlich Software auf einer grafischen Ebene beobachtet. Die Beobachtung erzeugt ein Interaktionsprotokoll. Im zweiten Teil werden Interaktionsmuster definiert und im Interaktionsprotokoll erkannt. Interaktionsmuster sind Interaktionssequenzen, die eine beabsichtigte Handlung ausdrücken. Dieses Verfahren wird *GUI Usage Mining* genannt, das Benutzungsinformationen von Interaktionsprotokollen der grafische Benutzeroberfläche entdeckt. Bezugsinteraktionsmuster werden anfänglich von menschlichen Kommentatoren entdeckt. Die daraus gewonnenen Erkenntnisse werden benutzt, um das unbearbeitete Interaktionsprotokoll aufzubereiten. Arbeitsvorgänge, welche sinnvolle Interaktionsabschnitte darstellen, werden durch vier Verfahren identifiziert. Das befähigt die Anwendung von vier Entdeckungsstrategien: *Sequential Pattern Mining*, *Graph Mining*, *Process Mining* und eine auf N-Grammen basierende Entdeckung. Diese sind mit der Hilfe von fremden Programmbibliotheken umgesetzt. In der Auswertung werden Paare von Benutzern zu Anwendungen analysiert. Die Interaktionsmuster von den vier Entdeckungsstrategien werden von den entsprechenden Teilnehmern beurteilt. Die Auswertung deutet an, dass das auf N-Grammen basierende Verfahren die akzeptiertesten Interaktionsmuster entdeckt.

Contents

1 Graphical Software Mining **7**
1.1 Introduction . . . 7
1.2 Problem Statement . . . 7
1.3 Background Research . . . 9
1.4 Related Work . . . 10
1.5 Approach . . . 11
1.5.1 Interaction Initiation . . . 11
1.5.2 Program Identification . . . 12
1.5.3 User Identification . . . 12
1.5.4 GUI Element of Interest . . . 13
1.5.5 GUI Element Identification . . . 14
1.5.6 GUI Element Alignment . . . 15
1.5.7 GUI Asynchrony . . . 16
1.5.8 Privacy Issues . . . 17
1.6 Implementation . . . 18
1.7 Evaluation . . . 20
1.7.1 Example Participant . . . 24
1.8 Conclusion . . . 33
1.8.1 Summary . . . 33
1.8.2 Discussion . . . 34
1.8.3 Motivation . . . 35

2 GUI Usage Mining **36**
2.1 Introduction . . . 36
2.2 Problem Statement . . . 36
2.3 Background Research . . . 37
2.3.1 Patterns . . . 37
2.3.2 Web Mining . . . 38
2.3.3 Frequent Pattern Mining . . . 39
2.3.4 Process Mining . . . 40
2.4 Related Work . . . 41
2.4.1 Web Mining . . . 41
2.4.2 Graph Mining . . . 42
2.4.3 Process Mining . . . 42
2.4.4 Bayesian User Modeling . . . 43
2.5 Approach . . . 43
2.5.1 Reference Patterns . . . 43
2.5.2 Preprocessing . . . 46
2.5.3 Strategy 1: Sequential Pattern Mining . . . 49
2.5.4 Strategy 2: Graph Mining . . . 50
2.5.5 Strategy 3: Process Mining . . . 51
2.5.6 Strategy 4: N-Gram Based . . . 52
2.6 Implementation . . . 53
2.6.1 Strategy 1: Sequential Pattern Mining . . . 53
2.6.2 Strategy 2: Graph Mining . . . 54
2.6.3 Strategy 3: Process Mining . . . 56
2.7 Evaluation . . . 57
2.7.1 User-Program Pairs . . . 57
2.7.2 Interaction Time . . . 60
2.7.3 Transaction Identification . . . 61
2.7.4 Pattern Mining Setup with Reference Pattern Analysis . . . 67
2.7.5 Pattern Analysis . . . 68
2.8 Conclusion . . . 78
2.8.1 Summary . . . 78
2.8.2 Discussion . . . 78
2.8.3 Outlook . . . 80

List of Figures

1 Alignment candidate generation based on hash code comparison . . . 16
2 Chart which shows the three groups of users depending on the distinct program usage . . . 20
3 Interaction distribution of used application softwares by participant 8 . . . 27
4 Gantt chart which shows application software usage over time of participant 8 . . . 29
5 Event distribution of participant 8 . . . 30
6 EOI control type distribution in Outlook of participant 8 . . . 30
7 Element of Interest distribution in Outlook of participant 8 . . . 31
8 Histogram of the ordered time spans between the interactions of participant 8 . . . 32
9 Crawl distribution in Outlook of participant 8 . . . 32
10 Taxonomy of Desktop Usage Mining . . . 37
11 Taxonomy of Web Mining [28, Figure 1] . . . 38
12 Ordered pattern score quantity of all strategies regarding to 3 or 2 rated patterns . . . 72
13 Pattern score chart which shows the quantity of scores . . . 74
14 Ordered average pattern score depending on n and k parameter of the k-skip-n-gram strategy (S4) . . . 74
15 Quantities of discovered patterns depending on different strategies (S) and transaction identification approaches (TI) . . . 75
16 Average pattern score depending on different strategies (S) and transaction identification approaches (TI) as well as n/k values . . . 76
17 Comparison of the four transaction identification approaches with an error chart ignoring the strategy . . . 77
18 Comparison of the four strategies with an error chart ignoring the transaction identification approaches and n/k values . . . 77

List of Tables

1 Services of the Windows Automation API . . . 18
2 Modules of the first pipeline . . . 19
3 Modules of the second pipeline . . . 19
4 The 9 observed participants on various Personal Computers (PCs) . . . 20
5 Version equivalence classes of the used application softwares . . . 23
6 A simplified snippet of the interaction log of participant 8 . . . 28
7 The 25 promising user-program pairs and their classifications sorted by distinct functional interactions . . . 59
8 The 25 user-program pairs with regard to their interaction time . . . 60
9 The 25 user-program pairs with regard to the first transaction identification approach (TI1) . . . 63
10 The 25 user-program pairs with regard to the second transaction identification approach (TI2) . . . 64
11 The 25 user-program pairs with regard to the third transaction identification approach (TI3) . . . 65
12 The 25 user-program pairs with regard to the fourth transaction identification approach (TI4) . . . 66
13 The 5 evaluators from the 25 user-program pair list . . . 68
14 The 25 user-program pairs with regard to the evaluation results for each strategy (S), transaction identification approach (TI) and n/k value . . . 73

List of Listings

1 Sequence database in the SPMF input file format . . . 53
2 Example result statistics of the VMSP algorithm . . . 54
3 VMSP result in the SPMF output file format . . . 54
4 Serialized graph in GraphML format . . . 55
5 Direct access to the ParSeMiS mining functionality . . . 55
6 The packages.xml file of the Pattern Abstractions package . . . 56
7 Direct access to the Pattern Abstractions plug-in functionality . . . 57

List of Definitions

1 Definition (Interaction) . . . 7
2 Definition (User) . . . 7
3 Definition (Program) . . . 8
4 Definition (Events) . . . 8
5 Definition (Observer Event) . . . 8
6 Definition (Process Event) . . . 8

7	Definition (Keyboard Event)	8
8	Definition (Mouse Event)	8
9	Definition (Crawl)	8
10	Definition (Element)	9
11	Definition (Identification Properties)	9
12	Definition (Parent Element)	9
13	Definition (Dynamic Properties)	9
14	Definition (Problem Definition I)	9
15	Definition (Eclipse Interaction)	10
16	Definition (Graphical Software Mining)	11
17	Definition (Observer Interaction)	12
18	Definition (Program Hash)	12
19	Definition (Program Hash Function)	12
20	Definition (Element of Interest)	13
21	Definition (Pattern)	37
22	Definition (Problem Definition II)	37
23	Definition (Desktop Usage Mining)	37

Introduction

This thesis is originated by the motivation of user assistance. User assistance helps users working with software. This includes manuals, tutorials, wizards and the containing text. However, this involves also helplines which are hotlines for PC issues. An online help doesn't involve interaction with an expert. This help is topic-oriented and usually included application softwares.

The offer of help is large, however few users actually use it. While books and hotlines can cost money, reading manuals, help texts and tutorials cost time. Furthermore, nobody wants to use additional energy to understand an application software for a small problem. The real tasks are expressed by users usually in a sentence like: "All I want to do is simply . . . ". This explanation is frequently heard by family members (usually experts) who have the duty to solve these daily problems. Because they are family members, they don't cost (a lot of) money. And because they are usually experts, they can solve the problems instantly. However, it's their lifetime and not actually their problems.

That's why a private project called HICon (an abbreviation for Human Interaction Concepts) was founded. The project is concerned about the question of how user assistance can help people working with the PC. Some thoughts include the usage of semantic meta-data, ontologies and natural language processing. However, the central problem seems to be the difficult access of the provided functionality. While application softwares implement useful functionality, the GUI is like a colorful curtain which complicates the functionality access. That's why experts exist who know the appropriate clicks to solve corresponding problems.

This consideration leads to the motivation of GUI assistance. A first idea was the implementation of a search engine. This attempt is based on the notice that GUI elements encapsulate functionality and at the same time are labeled with natural language. However, this demands a technology to receive necessary GUI element information from application softwares. If this would be possible, further user-centric considerations could be interesting. By simply adding an input device monitoring system, user and simultaneously GUI behavior could be observed. If such data would be acquired, knowledge discovery could mine interesting GUI usage insights.

Thus, this thesis asks the following leading question: Is is possible to observe users in their day-to-day work on the Desktop and can this give new insights about their GUI usage? In answering this central question this thesis is divided in two parts.

The first part **Graphical Software Mining** is concerned about the observation and capturing of interactions between users and application softwares. The problem statement defines a data scheme for storing interactions. In the background research is revealed what technology is used to receive GUI element information from application softwares. However, this technology is not new and used in the field of automated GUI testing. Some related papers give insights in the usage of that technology. The approach focuses on the realization of the observation: The entities user, application software (program) and GUI element have to be identified. GUI elements are further analyzed regarding to the interestingness of the user. The investigation results in an element of interest. Furthermore, the storage of GUI elements becomes a challenging task. Additionally, one has to face problems regarding GUI asynchrony and privacy issues. However, at the end 17759 interactions from 9 participants were collected in a so called interaction log. The evaluation gives detailed insights in that data.

The second part **GUI Usage Mining** considers the discovery of frequent interaction sequences — so called interaction patterns. But beforehand, the existence of patterns is argued with intentional actions. However, the background research reveals that patterns from other domains already exist. Web Usage Mining is a field of research which discovers patterns from data sample. The approach can be adopted to the GUI domain. Further investigations show that frequent pattern mining is able to discover interaction patterns. However, process mining seems also suitable for that approach. Thus, the related work covers web, graph and process mining papers. Additionally, a related paper about Bayesian user modeling is presented, too. The approach of this part utilizes four discovery strategies: Sequential pattern mining, graph mining, process mining and a novel n-gram based approach. However before that can happen, the interaction log has to be preprocessed. Human annotated reference patterns help to understand how patterns look like. This enabled the usage of four transaction identification approaches which are necessary for the first three strategies. These strategies are implemented with extern libraries. The evaluation selects 25 user-program pairs and analyses the interaction time and transaction identification. It is shown that reference patterns are rediscovered with the strategies. Finally, 1429 patterns are discovered and rated by evaluators. Statistics and some high rated patterns are presented.

Remarks

The observed graphical user interface is in the German language. That's why some examples and illustrations can contain German words. The study participants are also Germans. Thus, the user interface for pattern annotation and evaluation is labeled in German. Participants wrote pattern names in German, however they are translated in English.

1 Graphical Software Mining

Graphical Software Mining is a new term and implies using Software Mining exclusively on a graphical level. This enables the logging of interactions between users and application softwares. The result of this approach is an interaction log. This gives rise to new findings in the context of GUI usage.

1.1 Introduction

Nowadays complex systems hide useful insights in them selfs. Knowledge discovery extracts knowledge from such sources. If the source is man-made this process is called reverse engineering. This contains disassembling of the system, analyzing its parts and inner workings. The approach deduce the systems behavior without knowing the system at all. This is also true for software mining. In this case, software is the observed system which consists of artifacts. The discovered knowledge is usually expressed in a representative model of the software.

Software mining [25] investigates different levels in software. While the lowest level is concerned about statements and variables, the highest level analyses domain concepts and business rules. However, this thesis wants to add an additional level on top: the graphical level. Software is usually deployed with an application software which has a GUI; an interface to interact with the software. Graphical software mining investigates software exclusively on a graphical level. However, a GUI only comes to live if a user works with it. That's why the user is always an issue in the GUI interaction. While the user performs actions, the GUI reacts to them. Thus, we ask: Is it possible to observe and capture the interaction between users and application softwares in an interaction log?

This question can be answered with the *Accessibility* [58] technology. This mechanism gives access to GUI elements. On the other side, users work with input devices which can be monitored. However, working with the GUI is a time-critical real-time interaction process. Do the given technologies provide enough accuracy and performance to capture a correct and complete interaction log? A pipeline architecture, various caching mechanism and special algorithms are applied to face these problems. However, even the determination of the GUI element under the cursor becomes a non-trivial challenge. This comes from the fact that only few GUI developers care about the accessibility of their application softwares. Additionally, a data schemata of the observations has to be defined and filled. A very challenging task is the storage of persistent information from the volatile GUI system. Thus, an intelligent matching strategy has to be implemented. All in all, graphical software mining is more difficult than thought before.

The next sections are structured as follows: The **problem statement** section defines formally what is understood by the term interaction. The **background research** section reveals the necessary research to receive the ability of accessing the GUI information. The **related work** section covers similar approaches in the GUI automation, user tracking and software mining area. The **approach** section gives insights how observation is accomplished. The **implementation** section focuses on `WindowsAutomationAPI`, an Application Programming Interface (API) which implements the approach. In the **evaluation** section the collected data is briefly presented. At the end of this section a **conclusion** gives a summary, a discussion and motivates for the next section.

1.2 Problem Statement

Data about accessing the graphical user interface has to be acquired to mine it in the second part of the thesis. Currently there is no reference data about user-program interactions available. That's why this thesis will observe users in their daily work to gather real interaction data.

An interaction takes place between a user and an application software. The user initiates the interaction by performing an action with an input device (like keyboard or mouse) on a GUI element. The application software reacts with a (possibly hidden) state change. A crawl is a set of visible GUI elements describing the resulting state of the application software. An interaction observer examines both sides and stores interaction records in a database.

Formally, an interaction is defined as follows:

Definition 1 (Interaction)**.** $interaction := (id, timestamp, user, program, event, crawl, valid_crawl) \in Interactions$

Interactions (and subsequent entities) have a number for identification.

The time stamp tells when the interaction was initiated. It's in $datetime$ format and saves year, month, day, hour, minute and second. The *user*, *program*, *event* and *crawl* entries are complex objects described below.

Definition 2 (User)**.** $user := (id, machine_name) \in Users.$

Each Windows Operating System (OS) has a machine name (`NetBIOS` name of the local computer [61]) that is set by the user at setup time. For identification the machine name is stored. That's helpful for debugging and understanding the interactions in context of the user.

Definition 3 (Program). $program := (id, productName, productMajor, productMinor, productBuild, productVersion, originalFilename, fileDescription, fileMajor, fileMinor, fileBuild, fileVersion, language, companyName) \in SoftwarePrograms$

An application software (program) consists of attributes describing product name and its version, executable file name and its version, language and company name.

Definition 4 (Events). $Events = ObserverEvents \uplus ProcessEvents \uplus MouseEvents \uplus KeyboardEvents$

An event is the trigger for initiating an interaction. There are four disjoint types of events: observer, process, mouse and keyboard events.

Definition 5 (Observer Event). $observerEvent := (id, timeIncident) \in ObserverEvents$

$$timeIncident \in \{start, stop\}$$

The observer event occurs if the observer is started or stopped. The event exists to determine whenever the observation begins and ends.

Definition 6 (Process Event). $processEvent := (id, arguments, timeIncident, program) \in ProcessEvents$

$$\begin{aligned} timeIncident &\in \{start, stop\} \\ program &\in SoftwarePrograms \end{aligned}$$

The process event occurs if a process with a GUI is started or stopped. The event exists to determine whenever a certain program is started or stopped by the user. The arguments could be used to start the process with the same arguments again.

Definition 7 (Keyboard Event). $keyboardEvent := (id, key, style, alt, control, shift, hasKeyboardFocus) \in KeyboardEvents$

$$\begin{aligned} key &\in VirtualKeys \\ style &\in \{down, up\} \\ alt, control, shift &\in \{true, false\} \\ hasKeyboardFocus &\in Elements \end{aligned}$$

The keyboard event occurs if the user presses a key with a modifier (alt, control or shift). The $hasKeyboardFocus$ attribute gives indication of the focused GUI element. It can be interesting to know on which element the shortcut was performed.

Definition 8 (Mouse Event). $mouseEvent := (id, entity, style, data, elementFromPoint, accessibleObjectFromPoint, rankedFromPoint) \in MouseEvents$

$$\begin{aligned} entity &\in \{left, middle, right, xButton, hWheel, vWheel, move\} \\ style &\in \{none, click, doubleClick, down, up\} \\ elementFromPoint &\in Elements \\ accessibleObjectFromPoint &\in Elements \\ rankedFromPoint &\in Elements \end{aligned}$$

The mouse event occurs if the user operates with the mouse. A combination of entity and style reveals how the user controlled the mouse (e.g. left click or right double click).

Moreover the clicked GUI element has to be determined. In section 1.5.4 this thesis introduces the three methods $elementFromPoint$, $accessibleObjectFromPoint$ and $rankedFromPoint$ to receive the GUI element under the cursor.

Definition 9 (Crawl). $Crawl \subseteq ProgramElements \subseteq Elements$

$$crawl_valid \in \{true, false\}$$

A crawl is a set of GUI elements describing the resulting state of the application software. Every element that is for a certain moment visible for the user is contained in a crawl. One could compare it with a screenshot of the application software GUI state.

The flag $crawl_valid$ shows if the crawling was cleanly completed, therefore no other intervention occurred while crawling.

Definition 10 (Element). $element := (id, IdentificationProperties, index, parent, program, DynamicProperties) \in Elements$

Each GUI element has an id attribute for storing an identification number in the database. An element returned from a crawl doesn't have an instantiated id attribute. The underlying system doesn't assign a unique, reusable Identification number (ID) for identification. That's why we have to use a set of properties to compare elements and determine equality.

Definition 11 (Identification Properties). $IdentificationProperties = \{(key, value) \mid key \in Properties, value \in String \cup Integer\}$

Each property is a key value pair. The identification properties have only string or integer values.

Definition 12 (Parent Element). $parent \in Elements \cup \{null\}$

Each element has a parent. Elements having a null-valued parent are root elements. The result of this is a ordered hierarchical GUI tree. The siblings can be tell apart by an $index$ attribute. An element is part of a specific application software $program$.

Definition 13 (Dynamic Properties). $DynamicProperties \subseteq IdentificationProperties$

During runtime of an application software the property values of an element can change. We call such properties *dynamic properties*. The distinction is helpful because dynamic properties are variant and should not be used for identification.

The definitions define a data scheme. This scheme has to be filled with real life data. Thus, the problem is to extract and determine the necessary data. The interaction observer has to be implemented like a recorder. Changes of both participants in the interaction (user and application software) have to be detected. The data scheme helps to collect the data in a meaningful way.

To sum up, the problem can be defined as follows:

Definition 14 (Problem Definition I). *Given a GUI environment, observe the user who works with some application softwares. While the user triggers actions, the application softwares react to them. Hence, record both side and collect the insights in a database called interaction log.*

1.3 Background Research

Working with the PC seems to have no physical barriers. However for blind or visually impaired people the user experience drops rapidly because nearly every output of the PC is based on visualization. A screen reader [9] helps them to transform the visualized information to synthetic speech or a Braille display. But how does the screen reader access the information behind the GUI? The pixel based GUI seems to throw necessary information away when rendered to a pixel buffer.

A mechanism called *Accessibility* [58] gives other software (like screen readers) access to foreign application softwares' GUI elements. "Microsoft Active Accessibility (MSAA) was the earlier solution for making applications accessible" [68] in the Windows OS. However since 2005, "Microsoft UI Automation (UIA) is the new accessibility model (...)" [68].

Rob Haverty gives a brief overview in the paper *New Accessibility Model for Microsoft Windows and Cross Platform Development* [41] and states: "UIA provides programmatic access that allows automated tests to interact with the UI and allows assistive technology products to provide information about the user interface to their end users".

The White Framework [79, 78] is implemented on top of UIA. The API of White lets developers easily access the GUI without facing the complexity of UIA and windows messages. However, this thesis will implement an own library called *Windows Automation API*. The reason for refusing White is to have a more direct access to the underlying accessibility system. Thus, performance optimizations are easier to implement.

Ni Jin et al. [48] build a GUI automation framework called "AUILibrary". The paper presents a more technical view of building such a library. AUILibrary uses the old MSAA interface to access the GUI elements. They build this library "to search, identify varieties of controls, trigger mouse-clicking and keystroke events to simulate user's interactive behaviors" [48, p. III.]. It's related work because this thesis faces same problems. One problem is the finding of a unique UI element.

The author suggests to use a depth-first search method to prune the large Active Accessibility tree [48, III. B.]. The UI elements are abstracted to a common AUI class. They identify an AUI instance with a name, role, state, parent, window handle and child count [48, figure 2]. Problems with asynchrony is solved with so called Waiters. They wait for certain GUI events which can be used to synchronize the testing. However in contrast, the motivation of building such a library is to automate GUI interaction and provide GUI automation testing.

The following papers reuse UIA and build GUI models with the objective to support GUI automation. In contrast, Integrated Development Environments (IDEs) log usage information for interface improvements.

Atif Memon et al. [53] shows how a program automatically traverses a software's GUI and construct a GUI model. This approach is called "GUI Ripping". It's a reverse engineering process where the GUI model is constructed by executing the GUI program. The goal is to automatically generate test cases from a model. However, the interesting point in context of this thesis is the generation of the model. This model contains both "the structure and execution behavior of the GUI" [53, section 1]. An event-flow graph depicts the events flowing from one component to another. An integration tree is composed of windows where an edge exists if one window opens another. The work presents an efficient algorithm to extract a GUI model without having the source code of the application software. While traversing the GUI to form a GUI element tree, executable widgets are invoked by emulating a left-click mouse action. A hook mechanism raises an event if a window opens. At the end the integrated tree is generated. This related work shows that it is possible to construct a GUI model by execution observation. However, the GUI is traversed automatically by an agent without taking a real user into account. Moreover, the paper is not interested in creating an interaction log (exemplified in definition 1).

Another fascinating approach is presented by Pekka Aho et al. [3]. The Murphy tools "(...) automatically extracting GUI models based on dynamic analysis of the GUI" [3, section 1]. The motivation comes from safety and security issues in application software. More software testing makes the application softwares more robust and secure. But unit tests are not enough: GUI tests become more and more important. The paper implemented a tool called "Murphy [that] begins the testing of the GUI application already during the model extraction process" [3, section 1]. Compared with the GUI Ripping approach they both execute an application and observe the runtime behavior. To use Murphy a developer has to write a script that defines what and how many of the application under observation will be crawled. The running script automatically explores the application software and dynamically extracts a model. The model seems to be a graph of windows. An edge describes what element, existing in a source window, opens another target window [3, figure 2]. This demonstrates the possibility of observing application softwares and building a model. However, the motivation is still GUI testing in contrast to user behavior discovery.

Another way to model GUI behavior is presented by Xiaosong Li and Rick Mugridge [51]. The authors suggest the usage of petri nets [57] to model the behavior of the GUI. Simply put, a petri net is a graph with two kinds of nodes: places and transitions. "The PUIST model uses the Petri net structure and execution semantics to specify the dynamic behaviour of a GUI" [51, section 2.2]. GUI objects are separated in two classes, namely: action and base objects. Action objects are GUI events modeled as transitions in the petri net. Base objects are GUI elements (like dialogue boxes or windows) and represented as places. A base object is enabled if the place has a token. In context of a window enabled means displayed. The paper presents another model type to specify a GUI, but the authors are not concerned about automatic generation of such.

The above works have an application-centric view without taking real users into account. However the following references focus on user-centric data generation.

Eclipse [30] and NetBeans [22] are IDEs which implemented a tracking system that observes the usage of their platforms.

Eclipse's Usage Data Collector [7] monitors events (like activated, started, etc) and event producer (like view, editor, etc). Additionally, the runtime environment of the developer is tracked. Most of the information is IDE specific, like data about bundles, workbench, logs, etc. Each record stores a time stamp when the event occurred.

Formally, eclipse stores interactions as follows:

Definition 15 (Eclipse Interaction). $eclipse_interaction := (userId, what, bundleId, bundleVersion, description, time)$

$$what \in \{activated, started, executed, \dots\}$$
$$kind \in \{view, editor, command, \dots\}$$

The approach comes closer to the interaction formalization given in definition 1.

NetBeans' Usage Data Tracking "collect and analyze statistical information about high-level features NetBeans users use" [23]. The source reveals what kind of usage data is being collected, namely: IDE configuration, project type, web application framework, deployment, JDBC driver type, productivity feature, version control system and accessed help topics. It seems that the tracking system is more interested in the usage of certain features rather the usage of the GUI.

More scientifically, Ivan Benc et al. [8] collect web usage data by tracking users working with services. The motivation is discovering "users' habits, marketing strategies, generation of user profiles, and network performance optimization" [8, p. V.]. MidArc is a public information system mediator [8, section 3]. It mediates between users and services. The system collects usage records. They distinguish between general usage data and service specific usage data [8, IV. A.]. General usage data contains, among others, the service call time stamp. Specific usage data contains the call parameters and output of services. The system tracks usage data by building usage-collecting filters. These filters can be configured to store usage records. Thus, specific input or output data can be stored for analysis. The paper gives another view by collecting data from the web. The MidArc system is similar to the own proposed interaction observer approach. They are located between two entities and track the usage of the functionality requester and provider.

1.5 Approach

The previous references can be summarized under the term *Software Mining* [50]. This field of research combines data mining and reverse engineering on software. Useful knowledge about software artifacts (like source code, program states and structure) are mined. Software Mining can be classified in three types of analysis [16]: Static, dynamic and historic analysis. Dynamic analysis "happens when software is analyzed from the executive perspective" [16, section 1]. The mentioned papers about GUI automation above analyzed the application softwares while executing their graphical user interface.

This thesis extends Software Mining to coin a new term "Graphical Software Mining" and gives an informal definition:

Definition 16 (Graphical Software Mining). *Graphical Software Mining is the process of mining software exclusively on a graphical level. Dynamic analyses observe the graphical user interface of an application software at runtime by reverse engineer the structural and behavioral essence of the GUI. The result is a model describing the GUI for a specific purpose. Furthermore, the term also includes observing users in the role of interaction triggers.*

The approach observes users working with application softwares. The observer is the entity which is placed between user and application software. The problem statement (section 1.2) defined *what* data has to be collected. The challenge is to obtain the necessary data by observing user actions and the related GUI behavior.

The following sections explain how this thesis overcome these challenges. In particular, the **Interaction Initiation** section explains how the observer detects interactions initiated by the user. The **Program Identification** section defines how application softwares are identified whereas the **User Identification** section defines how users are identified. The **GUI Element of Interest** section gives a definition and covers how the element of interest is ascertained. The **GUI Element Identification** section gives insights how elements are identified between and during runtime. The **GUI Element Alignment** section focuses on the gathering of distinct elements. The **GUI Asynchrony** section reveals challenges with the asynchrony of the GUI. Finally, the **Privacy Issues** section covers issues on user's protected information.

1.5.1 Interaction Initiation

First, the observation has to detect whenever an interaction is initiated by the user. Three possibilities are examined: the user (1) starts or stops a process, (2) uses the keyboard input device, (3) uses the mouse input device.

(1) The observer detects an opened or closed process by looking in constant time intervals at the set of running processes of the OS. The difference of two consecutive observations ($Past$ and $Present$) reveals opened and closed processes. Mathematically, we can express the difference with sets:

$$Opened = Present \setminus Past$$
$$Closed = Past \setminus Present$$

However, this would return every opened or closed process. Unfortunately, the OS opens and closes a lot of processes in the background which haven't a GUI. The observer has to distinguish processes with and without a GUI to initiate only an interaction if the application software has a graphical front-end. That's why the observer waits for an opened window that is associated with a process. Once a window opens for a recently opened process an interaction is initiated.

(2) The observer detects keystrokes by using a global hook on the keyboard. But the observer should not be a keylogger [24] which records every key struck on the keyboard. Therefore, only keystrokes with modifier keys (control, alt or shift) are monitored. These key combinations are called shortcuts. Keyboard shortcuts are modifier keys in conjunction with other keys. For example, a typical shortcut is copy: *Ctrl + C*.

It is possible to hook on two types of events: down events (the key is pressed) or up events (the key is released). The observer only monitors down events, because consecutive up events would initiate a useless interaction and encourage invalid crawls (see GUI Asynchrony section 1.5.7 (c)).

(3) The observer detects mouse clicks by using a global hook on the mouse. A mouse has two buttons (left and right) and sometimes a wheel that can be used for scrolling (horizontal and/or vertical) and clicking (middle button). Moreover, x buttons are extended buttons for the mouse. In particular, "with Windows 2000, Microsoft is introducing support for the Microsoft IntelliMouse Explorer, which is a mouse with five buttons. The two new mouse buttons (XBUTTON1 and XBUTTON2) provide backward/forward navigation" [67, XButton1]. A user can use these buttons by pressing them (down event), releasing them (up event), making a click or double click. To keep things simple, the observer detects a click or double click from left, middle, right and x button. Mathematically, the possible pairs are in the cross product of the following sets:

$$\{reft, middle, right, xbutton\} \times \{click, doubleclick\}$$

For the sake of completeness, the observer itself initiates an interaction whenever the observation begins or ends. In this case, the $program$ and $crawl$ attribute is not set (null value assigned) because no application software is involved in this special interaction. The $crawl_valid$ attribute has the default value $true$. According to the interaction definition 1:

Definition 17 (Observer Interaction). $observer_interaction := (id, timestamp, user, null, observerEvent, null, true)$

For later transaction identification it is crucial to know when the observation begun and ended.

1.5.2 Program Identification

Each application software runs in a process. A process is "a set of instructions currently being processed by the computer processor" [44, (1.)]. Each process has an ID, in particular a Process Identification number (PID). The OS assigns PIDs arbitrary, hence same application softwares obtain different PIDs. We call such information *volatile* because these properties are only valid in a certain time period. To identify an application software among different users, one has to make such properties *persistent*.

An executable file is the origin of every process. To give the application software an unambiguous ID, a hash is calculated by taking the bytes of the executable file. This is called a "program hash".

Definition 18 (Program Hash). *A program hash is calculated by hashing the bytes of the executable file of an application software. The used hash function is Secure Hash Algorithm (SHA) with 512 Bits.*

Formally, the hash function maps a sequence of n bytes of arbitrary size to a fixed size SHA-512 hash.

Definition 19 (Program Hash Function). $f_{ph} : (b_i)_{i=1}^{n} \longrightarrow SHA\text{-}512, \quad b_i \in Bytes$

Application softwares originating in same executable files get same IDs. However, if developers release a new version of the application software it cannot be ruled out that the executable file also changes. That means different versions can cause different program hashes.

1.5.3 User Identification

The user ID is a hash of the following string concatenations:

- The CPU's
  - unique ID, or if not available
  - processor ID, or if not available
  - name, or if not available
  - manufacturer.
- The BIOS'
  - identification code,
  - serial number and
  - manufacturer.
- The Mainboard's
  - model,
  - name,
  - serial number and
  - manufacturer.
- The disk dive's
  - model,
  - signature,
  - total heads and
  - manufacturer.
- The video controller's
  - driver version and
  - name.
- The network adapter's
  - mac address.
- The personal computer's
  - machine name.

Additionally, the machine name is considered separately to have a human-readable identification. This was desired to debug the interaction observer. If a participant complains about errors an identification of a captured interaction stream was possible. Moreover, in the data evaluation the interactions could be analyzed in context of the known user.

1.5.4 GUI Element of Interest

Every time users interact with the GUI they focus attention on one GUI element. We call this control Element of Interest (EOI).

Definition 20 (Element of Interest). *The element of interest is a GUI element the user interacts with in a certain moment. For instance, an EOI is a clicked button, a text field the user enters text or a hovered menu item.*

In our case an interaction can be made with (1) the keyboard or (2) the mouse.

(1) The keyboard is used to send keystrokes to a focused element. Before the keyboard can be used the user has to choose which element receives the keystrokes. That's why there exists in the GUI always an element that has the keyboard focus. For example, a text field with a flashing cursor or a selected data item has currently the focus. Fortunately, the UIA Library implements a property called `HasKeyboardFocus` [60, UIA_HasKeyboardFocusPropertyId] that is true whenever the element has the keyboard focus. There can be only one focused element at the same time.

Logically, exactly one element that has the keyboard focus is the EOI in a keyboard initiated interaction.

$$\exists_{=1} e \in Crawl \quad HasKeyboardFocus(e) \equiv true \Leftrightarrow e \in ElementOfInterest$$

(2) The mouse is used to point to a hovered element. The element under the cursor in a certain moment (e.g. a mouse click) is the EOI in a mouse initiated interaction.

There are two methods to determine the EOI with a library call: (2.1) `AccessibleObjectFromPoint` [59] and (2.2) `ElementFromPoint` [66].

(2.1) The first function "retrieves the address of the IAccessible interface pointer for the object displayed at a specified point on the screen" [59]. This function is part of the earlier MSAA library. Some application softwares still support MSAA preferably, thus it will not do any harm to use this function. An additional call of the function `ElementFromIAccessible` has to be made to "retrieves a UI Automation element for the specified accessible object from a Microsoft Active Accessibility server" [65].

(2.2) The second function "retrieves the UI Automation element at the specified point on the desktop" [66]. This function is part of the UIA library. The state-of-the-art function returns in some cases incorrect elements. Because some application softwares don't implement the new UIA interface, (2.1) is used as a fallback.

Because there are some elements that disappear after being clicked both functions have to be called *before* the click is sent to the application software. This increases the chance to detect the correct EOI considerably.

Both functions return only a reference to the element. Later, the observer has to resolve the reference and crawl the desired property values. This has to be done in a period where the element still exists.

(2.3) A third auxiliary method `RankedFromPoint` is proposed by this thesis. This method compensates the errors made by (2.1) and (2.2). The approach keeps the two previous crawls of an application software in memory. We declare the nth crawl the current, $n-1$th the previous and $n-2$th the crawl before last. Mathematically, the following sets can be calculated:

$$\begin{aligned} Same &= n-1\text{th Crawl} \cap n-2\text{th Crawl} \\ Added &= n-1\text{th Crawl} \setminus n-2\text{th Crawl} \\ Removed &= n-2\text{th Crawl} \setminus n-1\text{th Crawl} \\ Candidates &= Same \cup Added \end{aligned}$$

Each element obtains an additional attribute called "level". The level expresses the instant of time the element is added to the $n-1$th crawl. For every element in $Same$ the level stays unchanged. For every element in $Added$ the level is set to the maximum level of the $n-2$th crawl increased by one. Every element in $Removed$ is not minded anymore. Only the elements in $Candidates$ are considered for further calculation.

In addition, each element has a $rectangle$ attribute that expresses the bounding rectangle of the element in physical screen coordinates. The approach calculates the intersection of the cursor location and all rectangle surfaces of the elements containing in $Candidates$. Every rectangle that intersects with the cursor location could be possibly clicked:

$$PossiblyClickedElements := \{e | e \in Candidates \wedge intersection(rectangle(e), cursor)\}$$

Finally, the elements in $PossiblyClickedElements$ are ranked with a multiple-criteria decision analysis. The reason is that no information exists about the order of elements in the z-space of the desktop. Four criteria are used to rank the elements where the top most element is decided to be the EOI.

(a) rectangle area
(b) level
(c) menu item control type
(d) foreground window belonging

(a) The more smaller the rectangle area is, the more likely the corresponding element is the EOI. However, this is not always the case: Smaller elements can underlay bigger elements. The intersection test can't distinguish elements on different levels. That's why other criteria are used.

(b) The newer the element is, the higher its level value is, the more likely it is the EOI. New elements have the habit to pop up and overlap older elements. Thus, in comparison between an old and a new element, the new element is more probable the EOI.

(c) Elements which are menu items are preferred. Menu items becoming visible and stay always on top of an application software. This circumstance justifies a preference of menu items.

(d) Elements which belong to the foreground window are preferred. This criterion helps to distinguish elements from different windows. The implicit levels of windows can be used to determine the foreground element and thereby the EOI. This becomes necessary if two or more windows overlap.

Last but not least, the outcomes of all three methods are compared. The agreement of the methods determines the EOI. Equivalent outcomes decide the final EOI of a mouse initiated interaction. For a better understanding a logical definition is given: The variable a is the element returned by the `AccessibleObjectFromPoint` function, the variable e is the element returned by the `ElementFromPoint` function and the variable r is the element returned by the `RankedFromPoint` function.

$$
\begin{aligned}
a \equiv e \equiv r \Leftrightarrow a \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
a \equiv e \Leftrightarrow a \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
e \equiv r \Leftrightarrow e \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
a \equiv r \Leftrightarrow a \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
r \not\equiv null \Leftrightarrow r \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
e \not\equiv null \Leftrightarrow e \in ElementOfInterest &\quad \text{else if} \quad ElementOfInterest = \emptyset \\
a \not\equiv null \Leftrightarrow a \in ElementOfInterest &
\end{aligned}
$$

1.5.5 GUI Element Identification

To tell apart unequal elements or to identify same elements an element needs a unique identification. This is necessary to rediscover elements between two crawls at distinct moments.

As already mentioned, each element can have a set of properties for identification. There are 111 properties [60] one can choose from. After some investigations the following subset of promising properties are used for identification:

- `Name` (string)
- `LegacyIAccessibleDescription` (string)
- `AccessKey` (string)
- `LegacyIAccessibleChildId` (int)
- `AutomationId` (string)
- `ClassName` (string)
- `ControlType` (int)
- `LocalizedControlType` (string)
- `LegacyIAccessibleRole` (int)
- `HelpText` (string)

Unfortunately, it is not possible to hash the values of these properties to form an ID because some property values change during runtime. That's why each element keeps track about its dynamic properties. For identification dynamic properties will be omitted. The next section covers in detail how elements are aligned with a database. This is done for identification between runtime.

The situation is different in case of identification during runtime. Every element has a runtime ID which is valid during runtime. The runtime ID "is an array of integers representing the identifier for an automation element" [60, UIA_RuntimeIdPropertyId]. During runtime elements can tell apart with the comparison of the runtime ID. Unfortunately, there are some elements that don't own a runtime ID. Therefore, this thesis proposes an approach to generate a runtime ID if not available.

The generated runtime ID is a concatenation of the runtime IDs of the element itself and its ancestors. If the runtime ID is available it is used for the generated one, otherwise the following attributes define a runtime ID:

- left, top, right, bottom attribute of its rectangle
- index attribute
- control type attribute

A mapping in $O(1)$ is possible if these attributes don't change. Now it makes sense to implement a cache where the keys are runtime IDs and the values are GUI elements. Every crawl of an application software is aligned with the GUI element cache. In case of a cache hit the element has not to be aligned in a time consuming process with the database. In case of a cache miss the element is added to the cache for later matching.

During runtime the observer has the possibility to detect dynamic properties. The cache helps to match same elements without the use of identification properties. In case of a cache hit the element in the cache and the element in the current crawl can be compared on an identification property level. If the cached value is different from the current value, the property value changed during runtime. These properties are marked as *dynamic*.

1.5.6 GUI Element Alignment

A database collects all elements examined by the observer. However, the aim is to store only distinct elements. In the previous section was explained how an element is identified. Because of dynamic properties and the GUI tree structure, the identification is more tricky. That's why a special alignment with the database is necessary.

Every element in the database obtains a unique database ID. An aligned element has an assigned database ID. The ID can be used to distinguish elements, once they are aligned with the database. Among users the same elements gets the same database IDs. From that time onwards the IDs can be used to communicate about elements.

Whenever the observer crawls an application software, all elements in the crawl are aligned with the database. The alignment algorithm is illustrated with the following pseudo code:

Algorithm 1: Alignment algorithm which assigns database IDs to local GUI elements of one application software

Data: ph is the program hash of the application software
Data: L is a local set of GUI elements from the the application software that has to be aligned with the database

1 $\mathfrak{S} := (S_i)_{i=1}^{n}$ ← calculate a ordered sequence of siblings from L;
2 p ← retrieve or create the application software by using ph;
3 D ← query every element of p from the database;
4 **for** $S_i \in \mathfrak{S}$ **do**
5     $S_i \leftarrow \{s | s \in S_i \cap CacheMiss\}$
6     $p_i \leftarrow parent(S_i)$;
7     $D_s \leftarrow \{d | d \in D \wedge parent(d) \equiv p_i\}$;
8   C ← calculate all hash code candidates between S_i and D_s;
9   **for** $C_i \in C$ **do**
10   **switch** $|C_i|$ **do**
11   **case** $= 0$
12   *The element is new and has to be stored in the database, hence a new unique database ID is assigned.*
13   **end**
14   **case** $= 1$
15   *The element exists in the database, hence the existing database ID is assigned.*
16   **end**
17   **case** > 1
18   *The $index$ attribute has to be used to determine the matching element.*
19   **end**
20   **endsw**
21   **end**
22 **end**

The algorithm expects a program hash and elements of a certain application software. After the execution every given element in L has an assigned database ID.

(Line 1) The elements span a tree. The tree structure is also used to identify an element. Thus, elements are compared level-wise in the tree. Therefore, the code generates a ordered list of siblings.

(Line 4) First, all root elements are aligned, then all children are aligned and so forth.

(Line 5) From this time on only siblings which had a cache miss are considered. If the cache already aligned the elements, it's not necessary to align them with the database.

(Line 7) Local siblings are compared with the siblings from the database. That's why all elements are selected which have the same parent. Because the algorithm starts at the top of the GUI tree, parents are aligned first.

(Line 8) At the beginning the GUI tree structure, afterwards the identification properties are used for comparison. In this line candidates are determined by exact matches. The algorithm is explained in detail below.

(Line 9-10) For every set of candidates the code looks at the size of the set.

(Line 17-18) It is possible that siblings have the exact same identification property values. At the very last instance the algorithm has to use the $index$ attribute to distinguish among them. Matching errors occur if elements are displaced.

In line 8, an exact match of property values is used to compare elements. For faster comparison a hash of property values is calculated. However, the dynamic properties prevent us from (pre-)calculating and simply comparing hashes. That's why dynamic properties has to be taken into account whenever a hash is calculated.

The following mathematical formalization shows how the hashes are calculated and the matching candidates are determined:

$$L := \text{set of local elements}$$
$$D := \text{set of database elements}$$
$$m := |L|$$
$$n := |D|$$

$$H = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{m,1} & h_{m,2} & \cdots & h_{m,n} \end{pmatrix}$$
$$h_{i,j} = hashcode(l_i, dynprop(d_j)) \quad l_i \in L, d_j \in D$$

$$\mathfrak{H} = \begin{pmatrix} \mathfrak{h}_1 \\ \mathfrak{h}_2 \\ \vdots \\ \mathfrak{h}_n \end{pmatrix}$$
$$\mathfrak{h}_i = hashcode(d_i, dynprop(d_i)) \quad d_i \in D$$

$$C_i = \{d_j | h_{i,j} \equiv \mathfrak{h}_j\} \quad 1 \le i \le m, 1 \le j \le n$$
$$C = \{C_i | 1 \le i \le m\}$$

Figure 1: Alignment candidate generation based on hash code comparison

The function $dynprop$ returns the dynamic properties of the given element in the database. The function $hashcode$ generates a hash from the identification properties, but ignores the given dynamic properties. Thus, only properties are used for comparison which seem to be persistent.

Every local element has to be matched against a database element. The database element decides what properties are used for comparison. That's why a matrix H has to be created where every hash code is generated with respect to the dynamic properties of the database elements. We don't know which local element matches against which database element. Therefore the matrix represents all possible matches.

$\mathfrak{H}$ is a vector of hash codes generated from the database elements with respect to their dynamic properties. Whenever the hash code $h_{i,j}$ is equivalent to the hash code $\mathfrak{h}_j$, the identification properties (ignoring dynamic properties) of l_i and d_j have an exact match. We call these database elements candidates for l_i and collect them in the set C_i. Finally, all C_i sets are combined in a set C which is returned in line 8.

1.5.7 GUI Asynchrony

The graphical user interface is an asynchronous system. After a user sends an event to the GUI (s)he can't know how or when the presentation reacts. That's why the observer has to include GUI asynchrony in its examination, too. This happens at three locations: (a) mouse event forwarding, (b) application software waiting and (c) clean crawling.

(a) Every time the mouse is used, an event is sent to an application software. There are some elements of interest which disappear after being clicked (e.g. a menu item) or other elements overlap the clicked element (e.g. a new window). An error of observation occurs if the observer would determine these EOIs *after* a click. Hence, the mouse event has to be intercepted first. Once the observer determined the EOIs (explained in section 1.5.4) the mouse event has to be forwarded. This process has to happen in a certain time range because the GUI reaction time grows.

(b) After an interaction is initiated, an application software's state is crawled. But how does the observer know *when* the application software accomplishes a state change? A constant waiting is inadvisable because some state changes are quick (e.g. loading a new tab) and some are slow (e.g. loading a file dialog). To determine the accomplishment of a state change, the thesis takes into account how the White framework handles waiting [77] in three different ways: Wait for (b1) process, (b2) window and (b3) cursor.

(b1) The function `WaitForInputIdle` "waits until the specified process has finished processing its initial input" [71]. On a process level the observer can determine if the process still calculates a state change. Thus, the observer waits until the function returns.

(b2) The UIA framework implements control patterns that "provide a way to categorize and expose a control's functionality (...) [69]. With the help of window patterns the observer can retrieve the current interaction state of a window. Hence, the observer waits until the window is responding. Additionally, by means of window patterns the observer can retrieve if the window waits for input. Thus, the observer waits until the window can receive new input. This indicates an accomplished state change.

(b3) Finally, the observer looks at the cursor. An hour glass cursor indicates that the foreground window still switches the internal state. Therefore the observer waits until the cursor is not an hour glass anymore.

(c) Whenever the observer crawls an application software, a named pipe is used to communicate all elements [85]. This can take an unpredictable amount of time. However, in the meantime another state change of the application software can happen. If this situation occurs the observer can't know what state is crawled. This is detected with simultaneous crawling and listening for new events. If a new event appears, while crawling is not finished, this event could change the state of the application software. In this case the observer marks the crawl *invalid*. Invalid crawls have to be enjoyed with some caution.

1.5.8 Privacy Issues

At the beginning of the project the interaction observer monitored every keystroke that was made by the user. Because the observer acted like a keylogger privacy issues were discussed. As a result the observer censors keyboard events that are sent to password input fields.

Fortunately, the UIA library implements an `IsPassword` property "that indicates whether the automation element contains protected content or a password" [60, UIA_IsPasswordPropertyId]. Moreover, the documentation suggests if an element both contains protected content and has the keyboard focus, "a client application should disable keyboard echoing or keyboard input feedback that may expose the user's protected information". However, the observer will not drop such keyboard events. The approach censors a keyboard event by replacing the key attribute with a constant default value (see definition 7).

Logically, if an element exists that has both `IsPassword` and `HasKeyboardFocus` set to true, then the keyboard event is censored.

$$\exists e \in Crawl \quad isPassword(e) \wedge hasKeyboardFocus(e) \Rightarrow censor(keyboardEvent)$$

Another privacy issue couldn't be implemented yet. It's the problem of privacy data expressed in GUI elements. A prominent example is an email subject. It's visible in common email programs as a data item representing an email. The data item names reveal email subjects. However, some GUI elements don't have private data because they are commonly used by all users. These elements are for example buttons, menu items and some alterable elements (e.g. check box, radio button etc).

In the case of password fields it's easy to detect private data. In the case of general GUI elements it is more difficult. There are two classes of GUI elements: (1) Private GUI elements and (2) common GUI elements. (1) The private elements seem to be dynamic elements which are completely different among users. The elements express user generated content. This can be contact lists, emails, files or written texts. (2) However, commonly used GUI element among users are elements of the application softwares. Thus, they don't contain private data.

If the observer can predict the class membership, private elements can be omitted. However, to predict the membership every GUI element has to be uploaded first. Only the elements used by many users indicate common GUI elements. This could be implemented with intersections of sets. Every user defines a set of used GUI elements with one specific application software. The intersection reveals commonly used GUI elements.

$$C = U_1 \cap U_2 \cap \cdots \cap U_n$$

The approaches above are implemented in an API called `WindowsAutomationAPI`, written in C#. This API reuses classes from a more general self-written API `AutomationAPI` which contains OS independent classes such as `Event`, `Service` and `ServiceConfiguration`. A service is a modular piece of functionality that takes care of exactly one problem. The `WindowsAutomationAPI` implements among others the services explained in table 1.

However, to mine the software on a graphical level, the thesis implements a project on top of the `WindowsAutomationAPI`. The core of this `SoftwareMining` project is a pipeline architecture. Every pipeline is a sequence of pipeline modules. A module receives a filled data container, processes the given data and adds computed results to the data container, thereby the next module can further processes the new data.

A `DualPipelineService` implements a pipeline system with two pipelines. The reason for separating the processing in two pipelines is a time critical handling of state changes by application softwares. Whenever the user initiates an interaction the application software's reaction will be observed. The state is represented with a set of visible elements (crawl). However, the crawling is time-consuming. If the state changes while crawling, the resulting crawl is marked *invalid*, because we can't know which state is actually crawled. The observer detects this by listening to new events. If a new event occurs, while the first pipeline is still processing, the outcome contains an invalid crawl.

The first pipeline (table 2) is used to process an incoming input event in real-time. The responsibility lies in the identification of the target application software, determination of the EOI and waiting for as well as crawling the application software. Table 2 explains the modules which are contained in the first pipeline in the sequence (from top to bottom). After the data passes through the first pipeline successfully, the second pipeline receives the data, therewith the first pipeline is free for new input events.

The second pipeline (table 3) continues processing the collected data in time-consuming tasks. These include the caching and alignment of elements as well as the determination of the ranked EOI and storage of the ultimate interaction. Table 3 explains the modules which are contained in the second pipeline in the sequence (from top to bottom).

The interaction observer runs in the background with a tray icon in the taskbar notification area. With the help of the `KeyboardInputService` and `MouseInputService`, input events are captured and inserted into the first pipeline.

Service	Description
`AutomationEventService`	This service uses the `AddAutomationEventHandler` method [63] to receive events if, for example, a window or a menu opens. A window open event is helpful to determine an opened GUI process (see section 1.5.1 (1)).
`CrawlService`	This service crawls a certain process and returns a `IUIAutomationElementArray` [64]. Additionally, the service can be used to crawl elements from point, handle or reference.
`DatabaseAlignmentService`	This service implements the alignment algorithm explained in section 1.5.6. Moreover, a version of the algorithm exists which aligns elements but doesn't store any.
`HardDriveService`	This service maps a path of an executable file to a program hash and vise versa. For faster access computed hashes are cached. Furthermore, the service extracts the necessary application software information.
`KeyboardInputService`	This service uses the `MouseKeyboardActivityMonitor` library [52] to hook globally on keyboard events. Thus, it is possible to detect interaction initiations explained in section 1.5.1. In addition, the `WindowsInput` library [54] is used to send keyboard events.
`MouseInputService`	This service uses the `MouseKeyboardActivityMonitor` library [52] to hook globally on mouse events. Thus, it is possible to detect interaction initiations explained in section 1.5.1. In addition, the `WindowsInput` library [54] is used to send mouse events.
`PcIdentifierService`	This service calculates the identifier of the user explained in section 1.5.3.
`ProcessPathService`	This service returns for a given PID the associated path of an executable file.
`TaskManagerService`	This service monitors running processes and fires an event if a GUI process opens or closes.

Table 1: Services of the Windows Automation API

Module	Description
`ProcessAddModule`	This module determines the process by looking at the event which initiated the interaction. In case of a keyboard event, the process associated with the foreground window is added. In case of a mouse event, the process associated with the window at cursor position is added. Additionally, the program hash is calculated and appended.
`ProcessFilterModule`	This module terminates the pipeline processing if given process names match with the observed process name. Thus, the module filters out undesired processes. This is especially the own process in which the interaction observer itself runs.
`ProcessInfoModule`	This module obtains the arguments by which the process is started.
`ResolveObservedFromPointModule`	This module crawls the references to the elements obtained by the two methods (2.1) `AccessibleObjectFromPoint` and (2.2) `ElementFromPoint` described in section 1.5.4.
`WaitModule`	This module waits for the state change of the application software explained in section 1.5.7.
`CrawlAddModule`	This module crawls the process of the application software.

Table 2: Modules of the first pipeline

Module	Description
`TransformCrawlModule`	This module transforms the flat crawled `IUIAutomationElementArray` into a GUI element tree.
`AddHiconRuntimeModule`	This module calculates the runtime ID explained in section 1.5.5.
`AutomationElementCacheModule`	This module caches elements by using the runtime ID. This is explained at the end of section 1.5.5.
`DatabaseAlignmentModule`	This module uses the `DatabaseAlignmentService` to align all element of the crawl.
`CrawlLastModule`	This module keeps the previous crawl of all application softwares in memory.
`HasKeyboardFocusModule`	This module determines the element which has the keyboard focus explained in section 1.5.4.
`RectangleTimeModule`	This module implements the third method (2.3) `RankedFromPoint` explained in section 1.5.4.
`PasswordModule`	This module censors keystrokes to password fields described in section 1.5.8.
`InteractionModule`	Finally, this module gathers all accumulated data to store an interaction record, described in definition 1, in the database.

Table 3: Modules of the second pipeline

This section gives an overview of the collected data by the interaction observer. After 70 days with 9 participants the observer collected 17759 interactions.

The 9 participants were family members and friends whose levels range from power over normal to weak users. Some participants use more then one PC. Thus, we have the following tree (Table 4) of 12 participant-PC pairs:

- Participant 1
  - PC 7C70
  - PC F52C
- Participant 2
  - PC 6FC4
- Participant 3
  - PC 5DE4
- Participant 4
  - PC 07BF
  - PC 85EC
- Participant 5
  - PC 8002
  - PC C7C1
- Participant 6
  - PC 0B8C
- Participant 7
  - PC 3CF1
- Participant 8
  - PC 8224
- Participant 9
  - PC 5727

Table 4: The 9 observed participants on various PCs

Figure 2 shows three groups of participants: weak, normal and power user (from bottom to top). The usage count of distinct application softwares determines the membership. No application software is counted twice, however different versions of the same application software result in multiple counting.

Weak users work with about 5 application softwares. The group of weak users exists because some of them only gave the interaction observer a try. After some problems the observer was deactivated or uninstalled. In particular, Participant 3 has only 512 MB RAM thus the PC was overcharged with the observer tool. Participant 9 uses primarily a macbook and if necessary a PC with the windows OS.

Normal users work with about 10 to 15 application softwares. More precisely, Participant 1 has two systems; one for some small tasks (PC 7C70) and one for the daily work (PC F52C). Participant 5 uses the PC for everyday matters, such as browsing and mail checking. Some tasks can be transferred to the mobile phone thus the desktop increasingly coming out of use.

Power users work with 30 to 50 application softwares. They use the PC for profession and/or digital work. In contrast to normal users, they have specialized application softwares for nonstandard tasks. That's why many various domain specific application softwares are used.

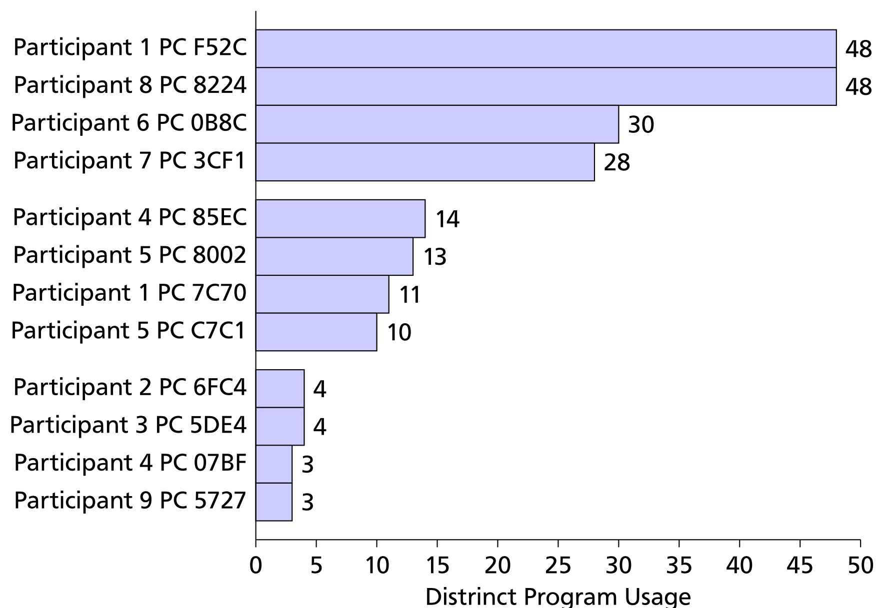


Figure 2: Chart which shows the three groups of users depending on the distinct program usage

The participants worked with 160 application softwares. However, some application softwares were counted several times due different versions of the same application. After merging different versions to one entity 105 distinct application softwares were counted. To be more specific, the table 5 shows for every application software its versions. The name of the equality class is the product name. Below the original filename in concatenation with the product version is presented.

This table is acquired as follows: Every application software, which is used by the participants, is considered. An application software is assigned to an equality class if the equality class member's average penalty is under or equal to a specified threshold. The penalty is the accumulated edit distance of the original filename and the file description. The threshold is set to 8.

Some versions occur multiple times due different program hashes but same meta data. Here we can see that the identifier is indeed persistent however too strict. More surprising is the fact that a small number of 9 participants have a high version distribution. Even standard office application softwares like Word, Excel or Outlook differ in version. The first entry "Kraken.Me" was a project of one participant who is a developer. Because of releasing and testing different versions occur. The second entry "StarMoney" is attributed to the fact that this application software is constantly updated. Updates from time to time, which change executable files, raise the versions, while no new application software is used.

Every user is related to an application software (program) if (s)he uses it. By forming tuples we count 247 user-program relationships. The application softwares are distinguished by their versions. The observer examined 4863 states resp. crawls of all application softwares. A total of 230963 distinct elements are identified. These elements are crawled, aligned and collected by the observer. However, the participants interacted with 4620 distinct elements. 161 of them are used with the keyboard and 4459 are clicked with the mouse.

These facts reveal that only 160 application softwares contain 230963 elements. However, due to mismatches some elements are stored multiple times. This comes from the fact that the identification is still too specific. Surprising is the fact that only 4620 elements from all 230963 elements are used by the 9 participants. That are approximately 97% of irrelevant elements. However, it must be said that the element matching algorithm is not stable. Thus, many elements could be counted twice. Moreover, there are many elements that are used for the GUI layout (e.g. Panel, Group, List, DataGrid etc) rather for interaction. Hence, the percentage is lower as 97% but still high. This comes from the fact that high-functionality applications (HFA) provide a lot of GUI elements and solutions while only a small subset is used.

The mouse is still the intuitive tool to point and click on graphical elements on the desktop. The shortcuts focus most of the time documents or text fields for text manipulation. While power user work with shortcuts excessive, normal and weak user only enter text.

- Kraken.Me
  - TKTracker.exe 1.3.1.0
  - TKTracker.exe 1.3.2.0
  - TKTracker.exe 1.3.6.0
  - TKTracker.exe 1.4.0.0
  - TKTracker.exe 1.2.1.0
- StarMoney
  - StarMoney.exe d45_83
  - StarMoney.exe d46_67
  - StarMoney.exe d46_6b
  - StarMoney.exe d46_6f
  - StarMoney.exe d46_70
- Java(TM) Platform SE 7 U60
  - javaw.exe 7.0.600.19
  - java.exe 7.0.600.19
  - jinstall.exe 7.0.710.14
  - javaw.exe 8.0.0.120
- Thunderbird
  - thunderbird.exe 31.0
  - thunderbird.exe 24.6.0
  - thunderbird.exe 31.1.2
  - thunderbird.exe 31.2.0
- Oracle VM VirtualBox
  - VirtualBox.exe 4.3.12.r93733
  - VirtualBox.exe 4.1.2.r73507
  - VirtualBox.exe 4.2.12.r84980
- Microsoft Office 2010
  - WinWord.exe 14.0.7125.5000
  - WinWord.exe 14.0.7134.5000
  - WinWord.exe 15.0.4641.1000
- Microsoft Office 2010
  - Excel.exe 14.0.7132.5000
  - Excel.exe 15.0.4641.1000
  - Excel.exe 15.0.4649.1000
- helppane.exe.mui
  - HelpPane.exe.mui 6.1.7600.16385
  - HelpPane.exe.mui 6.1.7600.16385
  - HelpPane.exe.mui 6.3.9600.16384
- sndvol.exe.mui
  - SndVol.exe.mui 6.3.9600.16384
  - SndVol.exe.mui 6.1.7600.16385
  - SndVol.exe.mui 6.3.9600.16384
- Notepad++
  - Notepad++.exe 6.5
  - Notepad++.exe 6.67
- Skype
  - Skype.exe 6.18
  - Skype.exe 6.20
- notepad.exe.mui
  - NOTEPAD.EXE.MUI 6.3.9600.16384
  - NOTEPAD.EXE.MUI 6.1.7601.17514
- Microsoft Outlook
  - Outlook.exe 14.0.7113.5000
  - Outlook.exe 15.0.4615.1000
- taskmgr.exe.mui
  - Taskmgr.exe.mui 6.3.9600.16384
  - taskmgr.exe.mui 6.1.7600.16385
- Microsoft® Visual Studio® 2013
  - vshost.exe 12.0.30723.0
  - vshost32.exe 12.0.30723.0
- Adobe Reader
  - AcroRd32.exe 11.0.8.4
  - AcroRd32.exe 11.0.9.29
- Microsoft OneNote
  - OneNote.exe 14.0.7107.5000
  - OneNote.exe 15.0.4631.1000
- cleanmgr.dll.mui
  - CLEANMGR.DLL.MUI 6.3.9600.17031
  - CLEANMGR.DLL.MUI 6.1.7600.16385
- iCloud
  - iCloudServices.exe 3.2.22.1
  - iCloud.exe 4.0.0.0
- WinRAR
  - WinRAR.exe 5.0.0
  - WinRAR.exe 5.1.0
- Microsoft Office 2010
  - POWERPNT.EXE 14.0.6009.1000
  - POWERPNT.EXE 15.0.4627.1000
- Microsoft® Windows® Operating System
  - dllhost.exe 6.3.9600.16384
  - dllhost.exe 6.1.7600.16385
- Dropbox
  - Dropbox.exe 2.10.27
  - Dropbox.exe 2.10.30
- Internet Explorer
  - IEXPLORE.EXE.MUI 11.00.9600.16428
  - IEXPLORE.EXE.MUI 11.00.9600.16428
- wmplayer.exe.mui
  - wmplayer.exe.mui 12.0.9600.16384
  - wmplayer.exe.mui 12.0.7600.16385
- Avira Product Family
  - IpmGui.exe 14.0.6.522
  - IpmGui.exe 14.0.7.266
- werfault.exe.mui
  - WerFault.exe.mui 6.3.9600.16384
  - WerFault.exe.mui 6.1.7600.16385
- soffice.exe
  - SOFFICE.EXE 3.04.9593
  - SOFFICE.EXE 4.00.9714
- rundll32.exe.mui
  - RUNDLL32.EXE.MUI 6.3.9600.16384
  - RUNDLL32.EXE.MUI 6.1.7600.16385
- Windows Installer - Unicode
  - msiexec.exe.mui 5.0.9600.16384
  - msiexec.exe.mui 5.0.7600.16385
- Avira Product Family
  - avgnt.exe 14.0.6.524
  - avgnt.exe 14.0.7.266
- avast! Browser Cleanup
  - BrowserCleanup.exe 9.0.2022.247
  - BrowserCleanup.exe 9.0.2022.267
- WinWein
  - WINWEIN.EXE 2.6.0.8
  - WINWEIN.EXE 2.6.1.0
- calc.exe.mui
  - CALC.EXE.MUI 6.3.9600.16384
  - CALC.EXE.MUI 6.1.7600.16385
- Adobe Acrobat
  - Acrobat.exe 11.0.07.79
  - Acrobat.exe 11.0.9.29
- VLC media player
  - vlc.exe 2,1,5,0
  - vlc.exe 2,0,7,0
- Microsoft Office 2013
  - VISIO.EXE 15.0.4641.1001
  - VISIO.EXE 15.0.4649.1000
- 7-Zip - 7zFM.exe
- Adobe Photoshop CS6 - Photoshop.exe
- Adobe Reader and Acrobat Manager - AdobeARM.exe
- Adobe® Flash® Player Installer/Uninstaller - FlashUtil.exe
- Amazon Music - Amazon Music.exe
- Amazon Music - setup.exe
- avast! Antivirus - AvastUi.exe
- cmd.exe.mui - Cmd.Exe.MUI
- openwith.exe.mui - OpenWith.exe.mui
- mstsc.exe.mui - mstsc.exe.mui
- mmc.exe.mui - mmc.exe.mui
- mspaint.exe.mui - MSPAINT.EXE.MUI
- xpsrchvw.exe.mui - xpsrchvw.exe.mui
- solitaire.exe.mui - solitaire.exe.mui
- dwm.exe.mui - dwm.exe.mui
- sdclt.exe.mui - sdclt.exe.mui
- recdisc.exe.mui - recdisc.exe.mui
- msdt.exe.mui - msdt.exe.mui
- wwahost.exe.mui - WWAHost.exe.mui
- systemsettings.exe.mui - SystemSettings.EXE.MUI
- xwizard.exe.mui - xwizard.exe.mui
- displayswitch.exe.mui - DisplaySwitch.exe.mui
- setup_wm.exe.mui - setup_wm.exe.mui
- Bluetooth Software - BTTray.exe
- Camtasia Studio - CamtasiaStudio.exe
- Camtasia Studio - CamPlay.exe
- CHIP Updater - CHIPUpdater.exe
- ColdCut - ColdCut.exe
- Cyberduck - Cyberduck.exe
- Cyberduck Updater - wyUpdate.exe
- Evernote® - Setup.exe
- Evernote® - Evernote.exe
- Firefox - firefox.exe
- Google Drive -
- iTunes - iTunes.exe
- Java(TM) Platform SE Auto Updater - jucheck.exe
- KeePass - KeePass.exe
- KeePass Password Safe 1.23 - KeePass.exe
- Lexware Info Service Assistent - LxUpdateManager.exe
- LiveUpdate - LiveUpdate.exe
- McAfee UI Container - McUICnt.exe
- Mendeley Desktop - MendeleyDesktop.exe

- Microsoft Office 2013 - ProtocolHandler.exe
- Microsoft® Windows® Operating System - glcnd.exe
- Microsoft® Windows® Operating System - MSASCUI.exe
- Microsoft® Windows® Operating System - PickerHost.exe
- Microsoft® Windows® Operating System - splwow64.exe
- MySQL Notifier - MySqlNotifier.exe
- MySQL Workbench - MySQLWorkbench.exe
- Norton Internet Security - ccSvcHst.exe
- PDF-XChange Viewer - PDFXCview.exe
- pgAdmin III - pgadmin3.exe
- Pidgin - pidgin.exe
- PuTTY suite - Pageant
- SourceTree - SourceTree.exe
- Steam Client Bootstrapper - steam.exe
- TeamViewer - TeamViewer.exe
- TeXnicCenter - TeXnicCenter.exe
- TortoiseSVN - TortoiseProc.exe
- VirtualDub - VirtualDub.exe
- WinSCP - winscp.exe
- WinWein System Info - WWSyInfo.exe

Table 5: Version equivalence classes of the used application softwares

1.7.1 Example Participant

This section gives a more detailed look on the data by using participant 8 as an example. Participant 8 made 3794 interactions with 48 application softwares. The participant was observed from September the 9th, 2014 through October the 28th, 2014, which was a period of approximately 53 days. The observation starts commonly at system boot time. However, every participant had the possibility to turn off the observation. This was done if bugs in the interaction observer prevent the work with the PC.

Figure 3 shows the quantity of interactions for every used application softwares. The quantity is illustrated with a bar and shows the 48 application softwares. They are ordered descending by the usage quantity. One special entry reflects the *observer interactions*. These interactions have no application software involved.

The first six entries shows what application softwares were important for participant 8 in this period of time. Outlook is an email client used for communication. The application software javaw stands actually for the Eclipse IDE. The Microsoft's Internet Explorer (iexplore) is used for web browsing. Microsoft's Powerpoint (powerpnt) and Word (winword) are slide and text processors. The PDF-XChange-Viewer (pdfxcview) is an alternative to the Adobe Reader. Thus, the main tasks of participant 8 on the PC are email communication, programming, information access (in the web and portable documents) and generation of digital content (presentations and documents).

The entries *sh* (Shell) and an empty entry (actually an unknown setup/uninstall program) reveal that there is not always beautiful meta data available. At the same time we see different versions of the same application software. These are excel (15.0.4649.1000 and 15.0.4641.1000), visio (15.0.4649.1000 and 15.0.4641.1001) and skype (6.18 and 6.20). This teaches us to consider version updates in the element and application software identification approach. A special case is iexplore which occurs twice, however with the same version 11.00.9600.16428 acquired from the meta data. This example shows that the program hash apparently distinguishes too strict.

Table 6 is a snippet of the collected interaction records (see definition 1) from participant 8. The table is based on the interaction definition 1.

Each interaction has a unique ID and the time stamp when the interaction occurred.

The user entry is actually an ID and the machine name. Because of privacy reasons we write Part.8 for participant 8. The interaction log contains all interactions from all users, however table 6 shows only interactions of participant 8.

The event table can contain four different event types. Thus, besides the type of the event five arguments are presented. In case of a mouse event, the arguments are mouse entity (Arg1), click style (Arg2), the ID of the element returned from the AccessibleObjectFromPoint method (Arg3), the ID of the element returned from the ElementFromPoint method (Arg4) and the ID of the element returned from the RankedFromPoint method (Arg5). In case of a keyboard event, the arguments are the pressed key (Arg1), a boolean value which states if the control modifier is pressed (Arg2), a boolean value which states if the shift modifier is pressed (Arg3), a boolean value which states if the alt modifier is pressed (Arg4) and the ID of the focused element (Arg5).

Finally, the abbreviate crawl ID and the valid crawl indication is on the right.

Investigation of the snippet of 35 interactions on October, the 16th gives some insights about the data. The data shows the interaction with the email client program Outlook.

As expected left clicks are the dominant mouse entity click style combinations. However, double clicks are also necessary in some cases. It attracts attention that frequently methods couldn't determine the EOI. In this case a *null* value is returned. While the AccessibleObjectFromPoint method doesn't work with Outlook, the ElementFromPoint method returns sometimes an EOI. That's why the novel RankedFromPoint method tries to compensate the result. However, even this approach fails sometimes and returns no EOI.

The typical shortcuts *Control + C* and *Control + V* are used for copy & paste interactions. However, the *Control + Return* shortcut appears which is used to invoke a procedure that sends an email. Similar to the mouse events, the EOI is sometimes not determined.

The crawls reveal the current state of Outlook. However, in specific cases the state is not crawlable. This is denoted with an *zero* crawl which is a zero hash. For example, between interaction 37702–37704 the state remains 0D5E. That's because the clicks and the paste-shortcut haven't change the set of visible GUI elements in Outlook.

Besides valid crawls there are also some invalid crawls. Invalid crawls occur if the application software is crawled while a new event is produced by the user. The Outlook application software presents a lot of elements. Thus, the crawling can take some time (some seconds). Because participant 8 is assigned to the power user group, it's not unusual that his/her interaction is fast. Hence, invalid crawls emerge.

Figure 4 shows a Gantt chart of application software usage by participant 8. It's only a snippet showing the first 7 days of observation. Each row is a different application software, the user interacted with. The blue bars represent interactions with one application software over time. Every day is 144 pixel long while a day has 1440 minutes. Thus, every pixel represents 10 minutes of the day. At the beginning of the study the interaction observer was unstable. Because the

participant has the possibility to turn off the observation, some holes exist in the data. It attracts attention that the first day contains no interaction. The first day is visible because an observer event was captured that day. Observer events don't have a corresponding application software. These charts give an initially overview over the acquired data.

Figure 8 demonstrates a histogram of the interaction time. The time span of every two consecutive interactions are plotted. The x-axis lists the interaction pairs, while the y-axis illustrates the time span in seconds. Because of long breaks the y-axis has a logarithmic scale. The time spans are descending ordered.

There are approximately 250 interaction pairs with a high temporal distance. However, from that point on the time span decreases linearly in logarithmic scale from about 100 seconds to 10 seconds. The most interactions take place in this period of time.

Figure 5 visualize the distribution of the events initiated by participant 8. As expected, mouse events are the most appearing ones with a quantity of 2121. 1957 of them are left clicks, 122 are left double clicks and only 42 are right clicks.

On the second place are keyboard events. This comes from the fact that at the beginning all keyboard events are observed. However, this unnecessary data crowded the database without any benefit. That's why only shortcuts are observed after an update of the interaction observer. Thus, 681 of the 1459 keyboard events are real shortcuts (at least one modifier is pressed).

Process events emerged 154 times. While 149 of them are open process events, only 5 of them are close process events. This comes from the fact that the observer couldn't notice closed processes. This gives room for some possibilities: The application softwares are never closed manually by the user, the OS closes the processes at shutdown or a bug in the observer prevents the realization. The total quantity is rather small if we consider that participant 8 works with 48 application softwares. However, 44 distinct application softwares are opened (and sometimes closed) in the 154 events.

Finally, observer events are the fewest events. They occur only 60 times. While 52 of them are observer start events, only 8 are observer stop events. This comes from the fact that the observer hasn't the chance to store a observer stop event, if the OS is shut down.

The following investigations focus exclusively on the interaction with the Outlook program. It's the most used application software of participant 8 with 1028 interactions which is 27% of the observation. Below the control type, GUI element and crawl distribution is examined.

Figure 6 illustrates the distribution of all EOI's control types in context of Outlook. The most frequent ones are explained below: The content of emails are usually visualized with the Document control type. That's why it is on the first place. This includes interactions with received and written emails. Furthermore Outlook consists of many buttons. They are used to invoke procedures as well as opening new content of Outlook. This promises that Outlook contains various functionalities. The table is usually used for displaying emails as data items. The data item control type occurs below the table control type. It can happen that in some circumstances a click to a data item is resolved to its table. An edit control type represents a text field. Text fields are used for entering small textual information (e.g. the email subject). Tree and list items representing usually user generated content like folder structures and contact lists. However, menu items are the building blocks of menu structures. Similar to buttons, the leafs of menu items invoke procedures as well as opening new content of Outlook. Thus, many menu item interactions indicates the existence of functionality.

Figure 7 demonstrates the 26 most frequently used EOIs in Outlook by participant 8. The most interactions are preformed with documents called "Unlabeled Message". This indicates that participant 8 writes or answers a lot of emails. It attracts attention that two documents with the same name exists. This is attributed to the too strict differentiation of EOIs. The main table "Table View" is on the third place. That's not surprising because emails are first listed in a table before they are opened. The EOI *Edit-24953* "Search Request" is used 32 times. Thus, participant 8 uses frequently a text field for searching emails or contacts. Some EOIs don't have a name or description. This is the case for *DataItem-57039*, *Custom-24806* as well as *TitleBar-24981*. Then the reproduction to related element in Outlook proves to be difficult. The buttons and menu items reveal what functionality is contained in Outlook: "calendar", "email", "minimize the window", "response", "close the window", "close all", "new appointment", "open", "response all" and "save & close". All in all, this distribution shows what EOIs are important for participant 8. However, there are 166 other EOIs which are used less then 5 times, but they are in sum 244 times interacted with. This reveals that participant 8 needs widespread EOIs and not a small subset of frequent ones.

Finally, figure 9 analyses the crawl distribution of Outlook used by participant 8. Unfortunately, this reveals that 277 times the state couldn't be detected. This can happen if the communication between Outlook and the crawler is interrupted. In this case the crawl contains erroneous information. However, the crawl 706E13 seems to be the most frequent one. This is a view to Outlook which occurred 71 times. In the case of *null* crawls no associated application software is available. This is always the case for observer events. Figure 5 revealed that they are 60 observer events. That's why there are 60 *null* crawls. However, there are not a couple of states in Outlook. 455 crawls with less then

6 appearances make 569 in total. This teaches us that many different sets of visible GUI elements are determined in Outlook. However, such information can help to detect the context of participant 8 in Outlook.

Application	Interaction Quantity
outlook 15.0.4615.1000	1028
javaw 8.0.0.120	611
iexplore 11.00.9600.16428	441
powerpnt 15.0.4627.1000	413
winword 15.0.4641.1000	326
pdfxcview 2.5	117
iexplore 11.00.9600.16428	71
amazon music 3.2.0.591	63
observer interactions	60
onenote 15.0.4631.1000	58
notepad 6.1.7601.17514	51
dllhost 6.1.7600.16385	45
acrord32 11.0.8.4	41
calc 6.1.7600.16385	40
keepass 2.27.0.0	39
mysqlworkbench 6.0.8.11354	35
camtasiastudio 8.4.3.1792	32
visio 15.0.4649.1000	30
visio 15.0.4641.1001	29
skype 6.18	29
teamviewer 9.0	27
pidgin 2.10.7	25
mendeleydesktop 1.12.1	23
dropbox 2.10.27	15
vlc 2,0,7,0	13
excel 15.0.4649.1000	13
tktracker 1.2.1.0	13
firefox 31.0	10
sh	10
msdt 6.1.7600.16385	10
sqlite database browser 2.0 b1	8
setup_wm 12.0.7600.16385	8
protocolhandler 15.0.4569.1503	7
wmplayer 12.0.7600.16385	7
excel 15.0.4641.1000	6
werfault 6.1.7600.16385	5
mysqlnotifier 1.1.5	5
msiexec 5.0.7600.16385	5
notepad++ 6.67	5
virtualbox 4.2.12.r84980	4
xpsrchvw 6.1.7600.16385	3
skype 6.20	2
bttray 6.5.1.2410	2
camplay 8.4.3.1792	2
rundll32 6.1.7600.16385	2
	2
texniccenter 2.0 Beta 2	1
tortoiseproc 1.8.1.24570	1
setup 3.6.0.671	1

0 1000

Interaction Quantity

Figure 3: Interaction distribution of used application softwares by participant 8

ID	Timestamp	User	Program	Event						Crawl	ValidCrawl
				Type	Arg1	Arg2	Arg3	Arg4	Arg5		
37701	16/10/2014 22:24:41	Part.8	Outlook.exe	KeyboardEvent	RETURN	True	False	False	53763	96CB...	True
37702	16/10/2014 22:27:52	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	24686	0D5E...	True
37703	16/10/2014 22:30:57	Part.8	Outlook.exe	KeyboardEvent	VK_V	True	False	False	53769	0D5E...	True
37704	16/10/2014 22:34:04	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	24687	0D5E...	True
37705	16/10/2014 22:37:17	Part.8	Outlook.exe	MouseEvent	Left	Click	null	146181	53763	50BC...	True
37706	16/10/2014 22:40:46	Part.8	Outlook.exe	MouseEvent	Left	Click	null	196835	196681	8E91...	True
37707	16/10/2014 22:41:07	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	25106	0000...	True
37708	16/10/2014 22:41:27	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37709	16/10/2014 22:42:03	Part.8	Outlook.exe	MouseEvent	Left	Click	null	53826	null	0000...	True
37710	16/10/2014 22:42:24	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37711	16/10/2014 22:42:45	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37712	16/10/2014 22:43:06	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37713	16/10/2014 22:43:42	Part.8	Outlook.exe	MouseEvent	Left	Click	null	146181	null	0000...	True
37714	16/10/2014 22:44:03	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37715	16/10/2014 22:52:11	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24659	null	4E18...	True
37716	16/10/2014 22:58:39	Part.8	Outlook.exe	MouseEvent	Left	Click	null	196838	24978	4E18...	True
37717	16/10/2014 23:03:49	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	24670	3FF1...	True
37718	16/10/2014 23:09:11	Part.8	Outlook.exe	MouseEvent	Left	Click	null	196838	24953	3FF1...	True
37719	16/10/2014 23:14:17	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	24670	3FF1...	False
37720	16/10/2014 23:19:37	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24670	24670	3FF1...	True
37721	16/10/2014 23:19:58	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	24670	0000...	False
37722	16/10/2014 23:20:34	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24670	null	0000...	True
37723	16/10/2014 23:25:47	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	3FF1...	False
37724	16/10/2014 23:26:24	Part.8	Outlook.exe	MouseEvent	Left	DoubleClick	null	24670	24670	0000...	True
37725	16/10/2014 23:31:29	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	3FF1...	False
37726	16/10/2014 23:36:50	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24670	24670	3FF1...	False
37727	16/10/2014 23:42:09	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24670	24670	3FF1...	True
37728	16/10/2014 23:42:16	Part.8	TKTracker.exe	MouseEvent	Left	Click	null	null	null	0000...	True
37729	16/10/2014 23:42:37	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	53761	0000...	False
37730	16/10/2014 23:43:13	Part.8	Outlook.exe	MouseEvent	Left	DoubleClick	null	57039	null	0000...	True
37731	16/10/2014 23:49:49	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	0FBA...	True
37732	16/10/2014 23:50:25	Part.8	Outlook.exe	MouseEvent	Left	Click	null	24659	53846	0000...	False
37733	16/10/2014 23:50:46	Part.8	Outlook.exe	KeyboardEvent	VK_C	True	False	False	null	0000...	True
37734	16/10/2014 23:51:07	Part.8	Outlook.exe	KeyboardEvent	VK_V	True	False	False	null	0000...	True
37735	16/10/2014 23:57:24	Part.8	Outlook.exe	MouseEvent	Left	Click	null	null	null	48B8...	True

Table 6: A simplified snippet of the interaction log of participant 8

Figure 4: Gantt chart which shows application software usage over time of participant 8

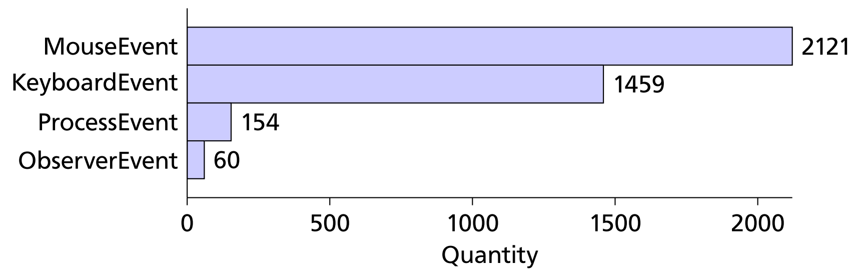


Figure 5: Event distribution of participant 8

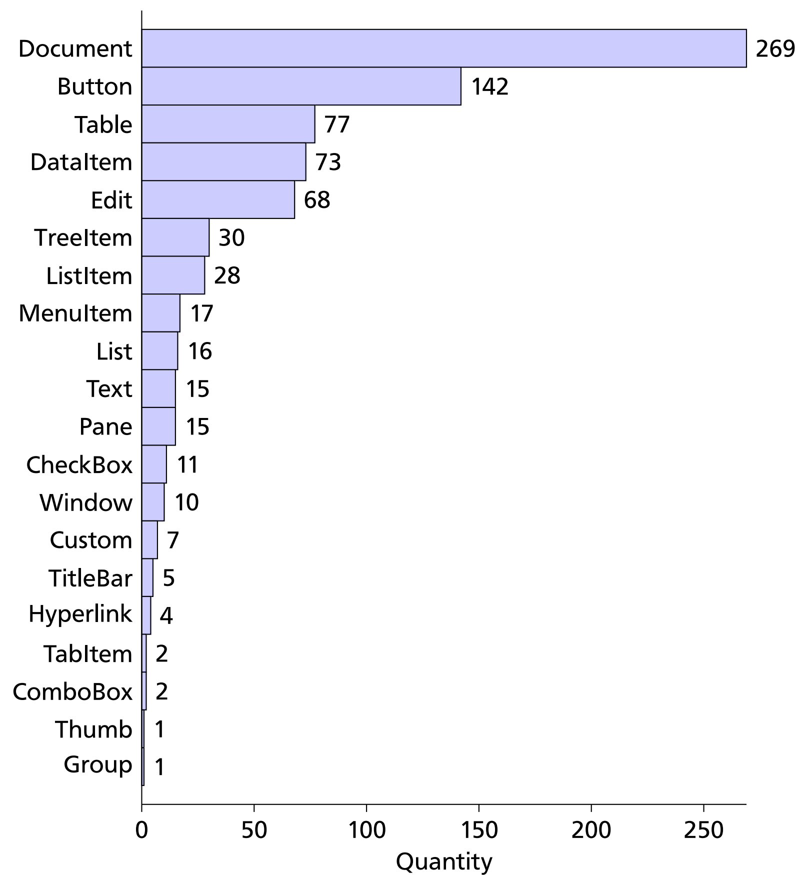


Figure 6: EOI control type distribution in Outlook of participant 8

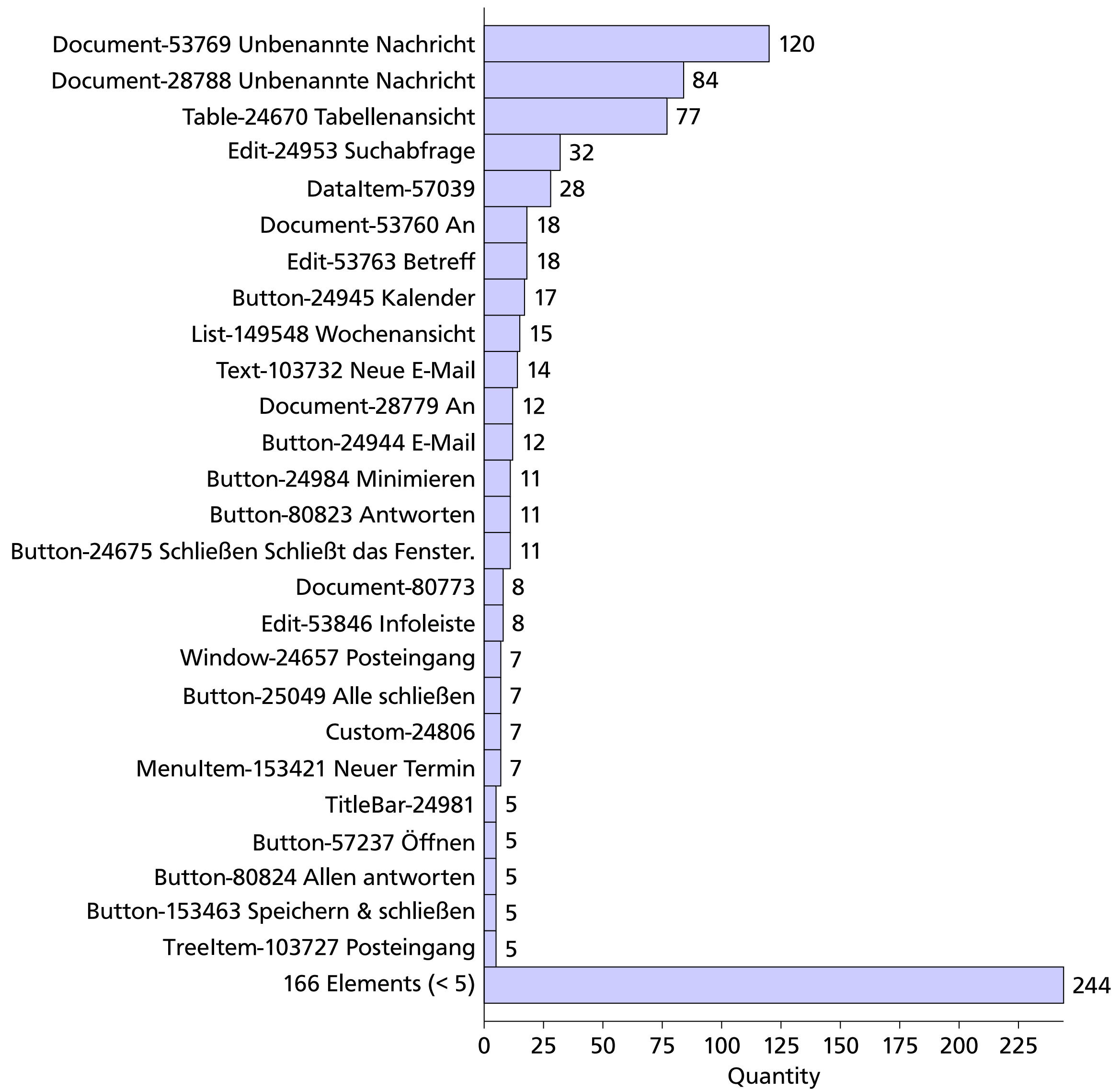


Figure 7: Element of Interest distribution in Outlook of participant 8

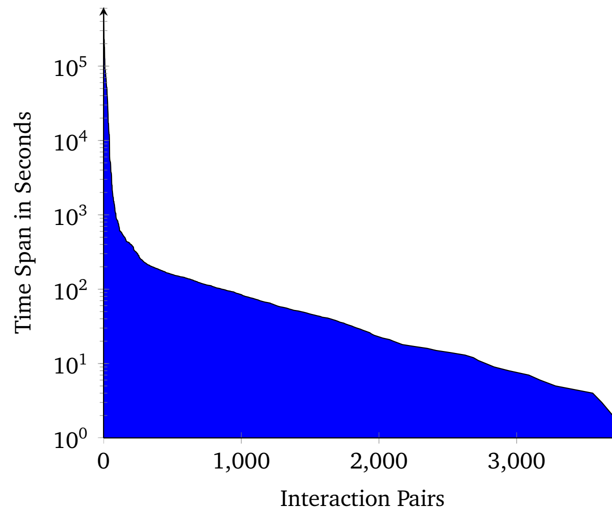


Figure 8: Histogram of the ordered time spans between the interactions of participant 8

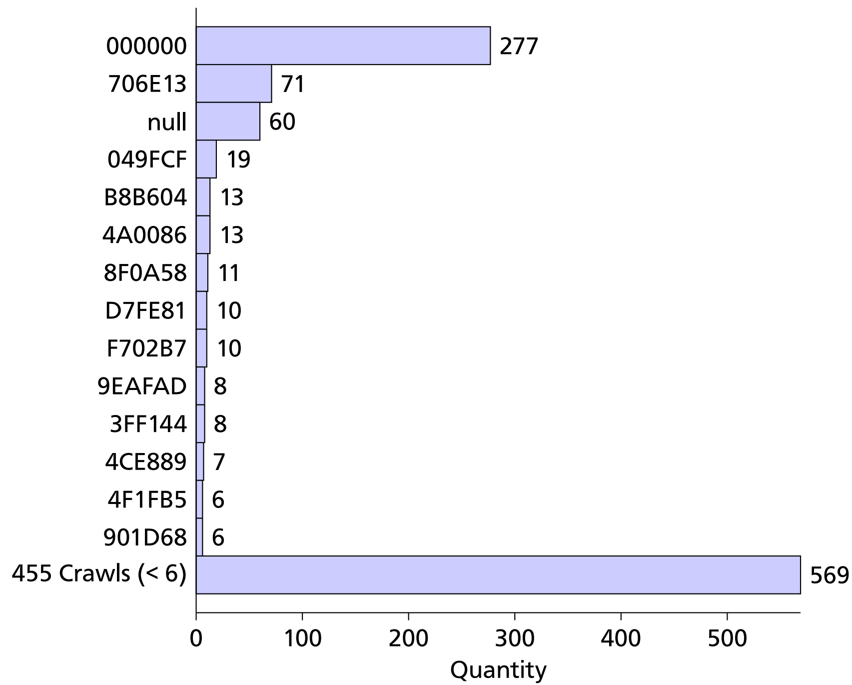


Figure 9: Crawl distribution in Outlook of participant 8

1.8 Conclusion

The conclusion summarizes this part and discusses the approach. Finally, a motivation bridges from the result of the first part to the second part of this thesis.

1.8.1 Summary

The topic was introduced with the motivation of reverse engineering. This is an important issue in the field of software mining. In contrast to traditional software mining, this part wanted to reverse engineer the graphical aspect of application softwares. However, it was also important to take the user into account. Thus, the main target was the observation of users working with application softwares. Based on the observation an interaction log was formed. However, first interactions had to be defined: While the user acts as an action performer, the application software reacts to that actions. With the help of a data scheme it was defined what data has to be collected.

The background research revealed what technology is necessary to receive the wanted data. Microsoft Active Accessibility is the answer to that problem. However, usually the technology is used for automated GUI testing. The related work demonstrated the so called "GUI Ripping" and GUI model extraction. These attempts came close to the vision of this part. However, the main focus was the support of GUI testing or specification. More closer came the usage tracking approaches of Eclipse and NetBeans. Related to this some approaches in the web exists, too. However, there is to the best of my knowledge no pure desktop interaction recording tool. That's why the approach defined the novel term Graphical Software Mining. The approach revealed how the necessary observations are implemented.

The first starting point was the realization of interaction initiations. In context of users, an interaction can be initiated with either a mouse or a keyboard input device. In case of the mouse, clicks with various buttons are monitored. In case of the keyboard, shortcuts are noticed.

Interactions are made with an application software. For identification among users the application software had to be uniquely identified. Because the GUI is a volatile system such identification had to be made persistent. This was accomplished with a program hash. Rather then the volatile PID, the program hash is a hash of the executable file.

Because users are an issue in the interaction, they were identified with a concatenation of system dependent strings. Additionally, the human readable machine name was stored.

Every application software consists of graphical elements. If users interacted with one of these elements, they are called elements of interest (EOI). It was explained how a EOI is determined in case of a keyboard interaction and in case of a mouse interaction. While a property `HasKeyboardFocus` made it easy to determine the EOI in case of a keyboard interaction, it was more difficult for the mouse interaction. Overall three methods were presented, while the third novel method tried to compensate errors. Finally, the outcome of all three methods are used to determine the final mouse EOI.

Unfortunately, graphical elements don't have a unique identification. That's why they had to be identified with properties between runtime. However, special dynamic properties made it more difficult to match equal elements. In contrast, during runtime the runtime ID can be used to match elements with a cache. This enabled the discovery of dynamic properties.

To gather all distinct elements a special alignment algorithm had to be implemented. This algorithm considers also dynamic properties. The inner workings were demonstrated with a pseudo-code and mathematical formalizations.

Because the GUI is an asynchronous system three issues had to be considered. The mouse event was forwarded after the mouse EOI was determined. This increases the chance for a correct EOI. Moreover, to crawl correct states of application softwares, the approach had to wait for the state changes. Finally, a crawl received the special mark *invalid* if the approach detects a state change while crawling.

Privacy issues were discussed. Keystrokes to password text fields were censored. Furthermore, private GUI elements were mentioned. However, only an idea of a solution was given.

The implementation focused on the inner workings of the self-written `WindowsAutomationAPI`. This API consists of services and a pipeline architecture with pipeline modules. The services implemented the functionality explained in the approach section. The pipeline modules processed incoming input events from the user to finally store an interaction record in a database.

The evaluation presented the observed data from 9 participants which installed the implementation (called interaction observer) on their PCs. Three groups of participants (weak, normal and power) were identified. The application softwares were clustered by their versions. Some statistical data was presented.

Participant 8 served as an example for detailed insights. This section looked at the collected interaction data from various angles: The quantity of interactions per application software, a snippet of the interaction log, a Gantt chart of usage, an interaction time histogram, the distribution of events, and detailed investigations with the Outlook application software.

1.8.2 Discussion

The following passages discuss which decisions turn out to be worse. Some suggestions for improvements are given. The section concludes with a lesson.

In the problem statement a data scheme is defined. However, there are many possible schemata with different focuses on the necessary data. Four types of events are specified. Maybe there are some more events that should be captured. This could be events produced by the application software itself. Users are not always the interaction triggers if we think about automatic update procedures or appointment messages. Thus, window or menu open events could be observed, too.

In the related work section there are papers presented which build GUI models. In doing so, they implemented similar approaches for the observation. Although they are focusing on automated GUI testing, the implementations and approaches could be adopted to the problem of recording the interactions. In fact, there are many GUI testing frameworks which support the recording of tests. This capture/playback approach could be reused to write desktop logs (similar to web logs). However, it is questionable if the inner workings of GUI testing frameworks can be adjusted that way.

The GUI identification is not appropriate enough. Although the program hash becomes a persistent alternative to the PID, it distinguishes too strict. From time to time updates for important security issues raise the quantity of different program hashes. As an example, the Online-Banking Software StarMoney had several updates in the observation period. Different program hashes result in different GUI elements, although they are equal. Most of the application softwares provide enough meta-data to distinguish them. However, some meta-data fields don't match exactly. That's why a fuzzy matching could help to identify programs based on meta-data. One field of the meta-data could still be the program hash. A look at table 5 reveals that there are many versions. Thus, the existence of different versions should be considered. These insights reveal that a fuzzy matching could solve some problems.

The user is identified with system dependent meta-data. However, if the system environment changes, the user ID changes too. One solution could be an account registration on a central server. Before the observation can start the user logs in with his/her account information. This approach could store additional information from the user. A social net of desktop users could emerge. One application could be the rating of used application softwares or the recommendation of application softwares by friends.

Determination of the element of interest (EOI) proved very difficult. While keyboard EOIs are determined by a special property, more work had to be done for mouse EOIs. However, the example log (table 6) suggests that this is still not enough. While method (2.1) and (2.2) are based on MSAA resp. UIA, method (2.3) is a novel algorithm to compensate the errors of the former two. This method appears more correct then the other ones, but still fails in some cases. Thus, it is still a challenge to simply determine the element under the cursor for a broad range of application softwares.

A very important part of the approach is the identification of equal GUI elements. With a set of 10 of 111 identification properties the GUI elements are aligned based on exact matches. To disambiguate GUI elements the GUI tree structure is used. This enables the usage of an index which distinguishes siblings. If the identification properties are insufficient this helps to tell apart elements on the same level. However, if the order changes or an element is inserted or removed, the matching becomes incorrect. A big problem are dynamic properties which change between runtime. However, keeping track of the dynamic properties was a good design decision. But this curses a too complex alignment algorithm which performs badly. Moreover, the proposed matching algorithm exposed to be too strict. That's why a fuzzy matching could solve the problem too.

The problem of GUI asynchrony was faced with three attempts. While the mouse event forwarding exposed to be a helpful technique, the application software waiting curses some problems. The determination of the current interaction state of a window can curse undesired behavior in some application softwares. In the case of the application software WinWein, the undesired behavior was a focus change. Thus, the application software was unusable. Besides, crawling of some other application softwares cursed also problems. That's why every contact with application softwares has to be enjoyed with some caution. Invalid crawls are a good control mechanism, however this information is not further used.

Privacy issues are solved in the context of password fields. However, if the interaction observer doesn't monitor every keystroke, the censoring becomes unnecessary. It turns out that the private GUI element problem is more important. The identification properties of the elements can contain human readable names and descriptions. While email subjects are less private, online-banking application softwares become a bigger problem. For example, from the application software StarMoney some account numbers were stored. That's why the problem of private GUI elements should solved to raise the acceptance of observation.

The evaluation observed 9 participants to acquire an interaction log. However, only 4 power user produced an acceptable quantity of data. This suggests the collection of few data points. Thus, further investigations will lack in expressiveness. Moreover, the gathered data is far from being complete and correct. This showed the snippet of participant 8. Completeness can be obtained if every EOI and crawl is determined and every event is monitored. Correctness can be obtained if every EOI is furthermore determined correct.

The time stamps seem to be odd. This could be a bug in the interaction observer. The time stamp should be the moment when an event enters, however the time stamp could be the moment when the interaction is stored in the database. That's why the periods are stretched. Unfortunately, this bug is discovered lately in the observation.

The lesson is clear: A lot of work still has to be done. To receive expressive statements from the interaction log, the completeness and correctness have to be very high. This improves the quality of the interaction log. However at the same time, privacy issues have to be solved to gain more acceptance in observation. This and a side-effect free interaction observer increases the amount of participants and consequently the amount of data.

1.8.3 Motivation

This section motivates further research in the next second part.

The first part *Graphical Software Mining* mined Software on a graphical level and includes the user in its observation. After 70 days 17759 interactions are observed. It must be emphasized that the interaction log is not a reflection of what users *think* they had done: It's an observation of 9 participants in their daily work with the PC. Hence, the interaction log represents what *actually* happened. However, noise exists because of reasons discussed above.

Nevertheless, first insights are gained by collecting statistical data. This is partly done in the evaluation section. A more detailed look gave the investigations on participant 8. The insights reveal frequently used application softwares and EOIs as well as frequent application software states. However, these insights are less expressive.

More expressive statements could be discovered if one looks at the structure of application softwares. With the help of the crawl we know the visible GUI elements of the application software. Additionally, the EOI reveals the element which causes that state. This gives rise to a structure of connected GUI elements. An edge would define a causality. For example, a click on a button curses the opening of a new window. Thus, the button (cause) and the elements of the window (effect) are related. This structure results in a navigation graph.

More interesting is the investigation of reappearing interaction sequences. Thanks to the EOI we know which GUI element was interacted with in an interaction. The sequence of EOIs are not arbitrary because humans are observed in their daily work. This suggests that there are meaningful interaction sequences in the interaction log. This investigation reveals more user-centric insights: The usage of application softwares and the manifestation of tasks in interactions. The next second part discovers such insights with GUI Usage Mining.

2 GUI Usage Mining

GUI Usage Mining is derived from Web Usage Mining. However, as the title suggests, focuses exclusively on GUI usage information. This necessary data is acquired from the interaction log of the previous section. On that data four pattern discovery strategies with four transaction identification approaches are applied and evaluated.

2.1 Introduction

Questions about the human usage of certain systems become more and more important. The answers help to understand the interaction between humans and systems. In particular, human-computer interaction [15] investigates the interactions between users and computer systems. Because computers have a high complexity, the interaction with them are not obvious. The human-computer interface is the point of intersection between the human and the digital world. Depending on the task different interfaces for communication are used. Insights about the usage of interfaces can improve them and opens the door for behavior analytics and assistance.

One of the todays commonly used interfaces is the graphical user interface (GUI). The user can, with the two simple input devices (keyboard and mouse), enter text and manipulate graphical elements on a screen. From the computer point of view the user is a producer of input events. However, the user performs meaningful interactions to accomplish tasks. Thus, an accomplishment of a task can be interpreted as a sequence of meaningful interactions. This is defined as an interaction pattern. Patterns are the first abstraction to understand the user-program interaction in the context of the GUI. It is assumed that the given interaction log from the previous part contains these patterns. Hence, we have to ask generally: Can we discover these patterns from a given interaction log?

This is the purpose of GUI Usage Mining. The idea is based on Web Usage Mining [75] — a related field of research. That's why we generalize the concept to the term Desktop Usage Mining. In the specific case of GUI Usage Mining, different mining strategies can be applied from other domains. Meaningful clusters in the interaction log, so called transactions, are the basis for mining algorithms. However, identifying these transactions turns out to be a difficult task. The shape of the transactions (separation and size) influences the outcome of the pattern discovery approaches. That's why a novel n-gram based discovery approach is applied, which doesn't dependent on transactions. Furthermore, one has to ask about the appropriate structure of the resulting patterns. Fortunately, given insights from other fields of research can be adopted. But which strategy is the most suitable one for discovering acceptable interaction patterns? This and other questions about the patterns are resolved in the evaluation.

The next sections are structured as follows: The **problem statement** section defines the meaning of patterns and establishes the new term "Desktop Usage Mining". The **background research** section investigates patterns and mining approaches in other domains. The **related work** section covers similar approaches in the fields of web, graph and process mining. The **approach** section gives insights in preprocessing and four pattern discovery strategies. The **implementation** section focuses on the usage of external libraries to implement the strategies. The **evaluation** section analyses the given interaction log and the resulting patterns. At the end of this section a **conclusion** gives a summary, a discussion and an outlook.

2.2 Problem Statement

G.E.M. Anscombe gives in her book *Intention* [6] a philosophical overview of the term. Whenever a user tries to solve a problem with the PC (s)he has to do an intentional action [6, §5]. Intentional actions "(...) are the actions to which a certain sense of the question 'Why?' is given application; (...)" [6, p.9]. Thus, if we ask a user why (s)he is doing some clicks and keystrokes, we hear answers like "I want that the word is double underlined" or "I want to empty the trash".

The PC forces the user to use the keyboard and/or the mouse as tools to express the intentional action. In a Windows, Icons, Menus and Pointer (WIMP) environment these devices seem to be enough to give the user the power to create an infinite variety of interactions. In the end, the user is a producer of an interaction stream. The environment forces the user to express the intentional action in a sequence of interactions. That's why the first example could be expressed with the following interactions:

- select the word (e.g. by double click on the word)
- left click on a button to double underline the word

The second example could be expressed in the following way:

- right click on the trash icon to open a context menu
- left click on the menu item "empty"
- left click on the button "yes" to confirm the action

The important question, *how* the intentional action is accomplished, can only be answered by an expert of the GUI environment and more importantly of the application software.

Every time a user wants to do the same intentional action (s)he is forced to use nearly the same sequence of interactions. We discover that there are sequences among users that reoccur over and over again. Such sequences are subsequently called "patterns".

The following informal definition explains what the thesis means by this term:

Definition 21 (Pattern). *A (graphical user interface interaction) pattern is a sequence of interactions that reoccur among users to express an intentional action. Patterns are caused by the GUI design of an application software: Users are forced to perform certain sequences of interactions which represent (sub-)tasks. The longer the sequence, the more tedious becomes the accomplishment of the related intentional action. The more frequent the sequence occurs, the more important is the related intentional action for the user's task. While atomic interactions are fine-grained mouse and keyboard usages, patterns give a more coarse-grained abstracted view on user activities.*

Mathematically, a pattern is simply a time-ordered sequence of interactions. $pattern := (i_j)_{j=1}^{n} \quad i_j \in Interactions, n \geq 1$ $\forall i_j, i_k \in pattern \quad j < k \Leftrightarrow i_j \prec i_k$*, where the predicate* $\prec$ *defines a time-ordering.*
If n is 1 the pattern is a trivial pattern, because only one interaction is the expression of an intentional action.

The main problem is to find and identify patterns. This can be formulated as follows:

Definition 22 (Problem Definition II). *Given a stream of interactions, made by one user with one application software, determine frequently interesting reappearing sequences which are the product of intentional actions (patterns).*

The problem can be classified under the term Desktop Usage Mining. This thesis presents a taxonomy of Desktop Usage Mining (figure 10) to give an overview and new promising fields of research. The task can be separated in functional and informative mining. On the functional side, Command-Line Interface (CLI) and GUI are both interfaces to invoke functionality. This work focuses exclusively on mining GUI usage data.

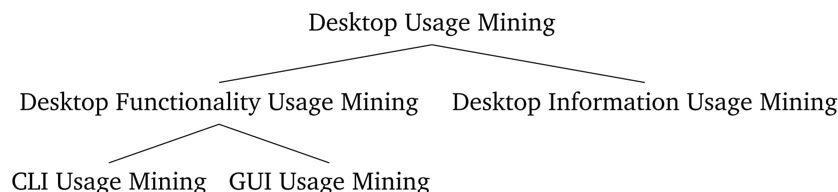


Figure 10: Taxonomy of Desktop Usage Mining

In addition, a brief definition of the term is given:

Definition 23 (Desktop Usage Mining). *The Desktop is an interface metaphor that presents GUI elements as if they would lying on a writing desk. Desktop Usage Mining is the automatic discovery of usage information from Desktop interaction logs. Desktop Information Usage Mining is interested in the accessed information (e.g. files, documents and media), while Desktop Functionality Usage Mining focuses on the functional aspect of the desktop (e.g. GUI workflows, patterns and commands). In particular, both CLI and GUI give access to functionality.*

However, it should be noted that Desktop Content Mining (similar to Web Content Mining) can also be termed. Desktop Content Mining is about collecting, identifying and indexing GUI elements. Thus, users can query GUI elements by Information Retrieval. Moreover, a search algorithm navigates through the GUI to the desired GUI element. In other words, it's a *Search Engine for Graphical User Interface* [43].

2.3 Background Research

The thesis undertakes background research in the following fields: First, patterns in other domains are investigated. Second, web mining (and in particular web usage mining) faces similar problems in finding patterns. Third, frequent pattern mining goes deeper in discovering patterns. Finally, process mining is another promising approach.

2.3.1 Patterns

The idea of finding certain patterns in human creation is not new. The architect Christopher Alexander is a pioneer in defining and finding patterns in cities, buildings and rooms which is for the first time proposed in one of his book *A Timeless Way of Building* [5]. For him "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" [4, p. x]. A pattern description has an example picture, a

name, the context for the pattern, a problem, a solution and finally other interlinked patterns. The author claims that if we use these patterns like words a pattern language evolves. Thus, we can easily communicate about working solutions with the help of patterns.

Computer scientists adopted the idea in the context of object-oriented software. They call them design patterns "that describes simple and elegant solutions to specific problems in object-oriented software design" [35, p. xi]. They describe patterns with a name, a problem, a solution and the consequences.

In architecture and resp. design the patterns are used primarily as a tool to build a complex system. In contrast, this thesis wants to use patterns to understand complex tasks of users in the GUI. Interacting with the GUI is not an effort of building something. However, there arise certain patterns for certain intentional actions. We realize the existence of patterns if we look at online help of an application software. Online help gives assistance how a specific application software is used. As an example we can read the help for the simple calculator in windows [55]. If a user has the intentional action to "convert values from one unit of measurement to another", the following interaction pattern has to be accomplished: (1.) Click the view menu, (2.) click the unit conversion submenu, (3.) select a type of unit, (4.) select a source type of unit, (5.) select a destination type of unit, (6.) enter the conversion value. Thus, interaction patterns are represented as a sequence.

Another way to represent patterns is proposed by the paper *Click Patterns: An Empirical Representation of Complex Query Intents* [29]. The authors want to understand the user who is interacting with a search engine. *Click Patterns* are probability distributions of click-through documents representing the likelihood of being clicked. The click patterns reveal the intent of the user that can be navigational, informational or a mixture of both (called semi-navigational). As a result click patterns describe a particular type of behavior of the user.

2.3.2 Web Mining

The following field of research undertake efforts to discover patterns in data samples.

Because of the explosive growth of information in the world wide web the field of research *Web Mining* [28] was founded. "Web mining can be broadly defined as the discovery and analysis of useful information from the World Wide Web" [28, section 1]. Web Mining itself splits up into two fields: Web content mining and Web usage mining.

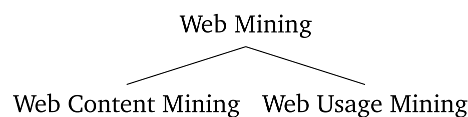


Figure 11: Taxonomy of Web Mining [28, Figure 1]

While web content mining automatically searches for information resources and make them accessible for the user, web usage mining discovers user access patterns from web servers. There are many similarities between web usage mining and desktop usage mining. That's why this thesis benefits from approaches in the web usage mining area.

It seems evident that server access logs can be used for mining useful information. While the most servers automatically store access information, the desktop lacks of such. That's why the first half of this thesis expends effort to observe the interaction of the user with the GUI to gather similar GUI access logs.

Besides "the life time value of customers, cross marketing strategies across products, and effectiveness of promotional campaigns", web usage mining can be used to "restructure a Web site to create a more effective organizational presence" [28, section 2.2]. The same motivation is applicable to the GUI where application softwares can be restructured depending on mined interaction patterns.

Pattern discovery tools are used to automatically discover association rules and sequential patterns from server access logs. Web usage miners are interested in user traversal path analysis in the context of websites. Like interlinked hypertext documents, GUI elements are also interlinked. As a result click paths emerges in the context of the desktop. For example, click paths are commonly used in help instructions and guidance with application softwares.

To mine raw access log data one has to preprocess it, for example "(...) developing techniques to clean/filter the raw data to eliminate outliers and/or irrelevant items, grouping individual page accesses into semantic units (i.e. transactions) (...)" [28, section 3]. Data cleaning removes irrelevant records to give an accurate reflection of the user. Transaction identification groups page references into logical units. The paper distinguishes between user sessions and transactions. Transactions split the user session depending on certain criteria.

The paper suggests performing path analysis by building a graph representation of the websites and find frequent traversal patterns. Association rules are proposed for finding associations in website access. Sequential patterns can be used to "find inter-transaction patterns such that the presence of a set of items is followed by another item in the time-stamp ordered transaction set" [28, section 3.2]. In addition, classification rules and clustering analysis are introduced.

A look in the paper *Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data* [75] gives a more detailed overview on web usage mining. This is defined as "the process of applying data mining techniques to the discovery of usage patterns from Web data" [75, section 1]. The browsing behavior of the user is tracked by using various data sources. This can range from web server logs over client-side data collection with Javascripts or Java applets to proxy traces from a proxy server.

Different data abstractions can be identified such as user, page view, click-stream, user session and server session. While it is difficult to identify a single individual accessing websites, on the desktop this becomes a trivial problem. Page views consists of files that has to be loaded when the user performs a single action (e.g. a mouse click). Crawls seem to be almost similar to page views. A click-stream is a sequence of page view requests. Click-streams can equally well be applied to interaction sequences. A user session is the click-stream of a single user accessing servers. In contrast, a server session is a user session for one specific server. Here again terms can be transfered to the desktop environment. Server sessions then called application software sessions.

After defining data abstractions the three phases are worked through: preprocessing, pattern discovery, and pattern analysis.

The first and most difficult task is to preprocess the information from various data sources. After user identification, the click-stream has to be separated in sessions. Incomplete click-streams occur because of cached page references. With the help of content preprocessing the content of websites can be taken into account. Thus, it is possible to limit the pattern discovery to certain websites. But dynamic page views make it difficult to determine the content. Structure preprocessing uses the structure created by hypertext links.

In the second phase patterns are discovered with the help of methods and algorithms from several fields. Statistical analysis is the first technique to get an overview of the data. However, this can't bring in-depth insights. Association rules gives exposure about websites that are associated (but not necessary referring) to each other. Clustering can compute usage clusters and page clusters. While usage clustering groups users with similar browsing behavior, page clustering groups websites with similar content. With the help of classification users can be mapped to predefined classes. Sequential pattern mining finds patterns that occur in a time-ordered set of sessions. Dependency modeling develops dependencies among various variables to model browsing behavior by using probabilistic learning techniques. Prominent techniques are Hidden Markov Models and Bayesian Belief Networks.

Finally, the third phase analyses patterns on the level of interestingness. Patterns are filtered out therewith useful patterns left over for a specific application. Visualization techniques can help to highlight appropriate patterns.

These three phases can easily be transfered to the desktop to do desktop usage mining. Motivated by web usage mining and the pattern discovery phase, this thesis investigates more background research in methods and algorithms from several fields.

2.3.3 Frequent Pattern Mining

The paper *Frequent pattern mining: current status and future directions* [40] gives a more detailed look at mining frequent pattern. There are three types of patterns: itemsets, subsequences and substructures. Frequent itemsets are sets of items that occur frequent in a transaction data set. Subsequences that appear frequent are called sequential pattern. Substructures are general forms like subgraphs or subtrees.

[40, section 2.5] explains the mining of sequential patterns. The author suggests the application to web clickstreams among others. That's why this approach seems very suitable for the collected interaction data. A sequence α is defined as an ordered list of itemsets. Items in an itemset occur at the same time, however itemsets in a sequence occur at distinct moments. A sequence $\alpha = \langle a_1, a_2, \dots, a_m \rangle$ is a subsequence of a sequence $\beta = \langle b_1, b_2, \dots, b_n \rangle$ if and only if $\exists i_1, i_2, \dots, i_m$ such that $1 \leq i_1 < i_2 < \dots < i_m \leq n$ and $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \dots, a_m \subseteq b_{i_m}$. That means there exists m ordered indices that range between 1 to n such that every itemset in α is a subset of the corresponding itemset in β. This permits holes and missing items in the subsequence but forbids a different ordering. Usually, the frequency of a subsequence is expressed with the support metric which counts the number of sequences in a sequence database that contains that subsequence. A minimal support (often written *min_sup*) is a threshold that specifies if a subsequence is frequent or not. Special *closed* sequences are sequences which don't contain in a supersequence which has the same support.

Besides that, [40, section 2.6] suggests the mining of structural patterns like graphs, trees and lattices. The idea is to find frequent subgraphs in a set of graphs. The interaction logs can be reinterpreted as graphs because interactions relates to each other with certain criteria. Thus, it is possible to build GUI element graphs or crawl graphs. Frequent subgraphs can then reveal frequent traversal paths or frequent application software state changes.

[40, Section 3] suggests the usage of an interestingness measurement because "such mining often generates a huge number of frequent patterns". Mining the interaction log will also find a lot of patterns that are frequent but however not interesting. With user-specified constraints the patterns can be checked

- at the start of mining (succinct constraint),
- during mining and pattern growth (anti-monotonic constraints) and
- during mining but if once satisfied not further in pattern growth (monotonic constraints).

Another way to calculate interestingness is to use a measurement of association or correlation relationship, such as for example support and confidence, lift, χ^2, cosine or all_confidence.

2.3.4 Process Mining

This thesis can also benefit from the following field of research: W.M.P. van der Aalst studies the research area *process mining* [2]. "Process mining aims at extracting information from event logs to capture the business process as it is being executed" [2, abstract]. In addition, it also focuses on causalities between activities. [2, Section 2] shows by an example how an explicit model is extracted from events. These events are real executions which are used to distill a structured process description. Such a description is called a process model and is represented usually as a petri net.

The interaction with the GUI can be interpreted as a process with various application softwares. The observed interaction data can be seen as an event log. Thus, methods and techniques to extract the behavior from the log can be used from process mining. In fact many challenges explained in [2, section 4] are similar:

Hidden tasks are tasks that are not recorded and therefore not present in the log and process model. This also happens while observing the interaction between the user and the GUI. However, it is possible, given a specific language, to detect these hidden tasks in the log. The author admits that this becomes more difficult for complex processes.

Moreover, process mining is interested in mining loops. Loops are tasks which are executed multiple times. In comparison, loops in interaction sequences can indicate patterns. It can be assumed that users also using loops, because a loop can be a jump back in their tasks. For example a characteristic loop could be a programming task, where the subtasks *edit code*, *compile*, *run*, *view console* repeat in a loop.

Time information, accomplished with a time stamp, can help to model time data, such as "flow times, waiting times, and processing times" [2, section 4.5]. Short time distances can help to determine causal relations between tasks. Fortunately, an interaction record has a time stamp and therefore additional time information can be used.

Process mining views the event log from different perspectives. One principal perspective is the control-flow perspective. This perspective reveal task orderings. However, several other perspectives are considered: organization perspective, information perspective, and application perspective. The organization perspective is interested in roles, groups and other organizational concerns (like e.g. responsibility, availability). Roles have a functional aspect, while groups have an organizational aspect. The information perspective handle production and control data. Control data can be variables that express process management. Production data don't depend on the process management and thus are only information objects. The application perspective looks closer at applications used in the tasks. This can be ordinary application softwares in a desktop environment. However, the control-flow perspective seems sufficient for analyzing the user's flow in the GUI.

Process mining also deals with noise. Noise is information that is incorrectly logged. That's why mining algorithms have to be robust in this case. As a result some algorithms allow a threshold value which decides between incorrect and exceptional behavior. This topic is also important for the gathered interaction stream. Because the interaction observer doesn't work accurately, noise appears in the data, too.

Moreover, incompleteness is related to noise. A log is incomplete if it doesn't have enough information for process modeling. Traces that occur very rare influence the derived process model and lead to incorrect models. The problem is that incompleteness gives scope for many possible models. Heuristics are used to restrict the possibilities.

The paper *Abstractions in Process Mining: A Taxonomy of Patterns* [10] abstracts given event logs by discovery of common patterns. The problem is that unstructured processes generate spaghetti-like process models. The reasons for this are activities living in isolation and not fully considered contexts. Thus, the approach finds patterns and replaces them by an abstracted entity in the event log.

[10, Section 3] defines a taxonomy of patterns. This taxonomy considers two types of patterns: Loops as tandem arrays [10, section 3.1] and sub-processes as conserved regions [10, section 3.2]. Loops are "repeated occurrence of an activity or subsequence of activities in the traces" [10, section 3.1]. However, more interesting for this thesis are sub-processes, because they fit to the definition of patterns: "similar regions (sequence of activities) common within a trace and/or across a set of traces in an event log signifies some set of common functionality accessed by the process" [10, section 3.2]. The paper distinguished three types of so called repeats. A *Maximal Repeat* (M) is a subsequence which occurs in a maximal pair, while a maximal pair is an identical pair of two sequences which would be unequal if they are extended on either side. A *Super Maximal Repeat* (SM) is a maximal repeat that is never contained in another maximal repeat as a subsequence. A *Near Super Maximal Repeat* (NSM) is a maximal repeat that is at least once not contained in another maximal repeat. These special repeats can help in finding choice constructs. In a set theoretical way, the sets of the three types are in the following relations:

$$SM \subseteq NSM \subseteq M$$

[10, Section 3.3] defines equivalence classes over repeats. Every repeat r consists of an alphabet $\Gamma(r)$. The alphabet is a set of symbols resp. activities which occur in the repeat. The equivalence class shows various repeats with the same alphabet.

The repeat alphabets are used in [10, Section 4] for subprocess abstraction. This is done by defining a partial ordering with the cover relation ($ra_1 \subset ra_2$) on the repeat alphabet. The abstraction approach generates a Hasse diagram on the partial ordering to receive a maximal repeat alphabet. These maximal elements can be used to abstract processes. Hence, "repeat alphabets under a maximal element can all be represented with the abstraction of the maximal element" [10, Section 4]. In the case of more then one maximal element the approach suggests extended joins on the maximal elements. This is useful for reducing abstract activities.

Finally, [10, section 7.1] presents the main idea of the approach: First, the approach discovers loop constructs (e.g. tandem arrays) and replace them with abstract activities, and second, common functionalities (e.g. maximal repeats) are discovered and abstracted.

2.4 Related Work

The following related works perform mining algorithms on a given log similar to the interaction log or build user models based on a stream of application software events. The works are analyzed in the following way: First, the text describes what they do. Second, it is examined how they achieve it. Third, results are presented. And finally, a statement explains why they are related to this thesis.

The sections cover three papers in the web mining context, insights into graph mining and two papers in the field of process mining. Finally, a paper is presented which creates Bayesian user models in the context of software assistance.

2.4.1 Web Mining

The related paper *Frequent Pattern Mining in Web Log Data* [47] demonstrates the usage of frequent pattern mining to receive three kinds of patterns from web log data: itemsets, sequences and tree patterns. [47, Section 5] explains how the mining is archived: The ItemsetCode algorithm is used for discovering frequent itemsets, the SM-Tree algorithm discovers frequent sequences and the PD-Tree algorithm finds tree-like patterns. But beforehand, [47, section 6] describes how the web log data is preprocessed. In the case of a web log, graphic and multimedia hits are removed from the log. Because the data is recorded on a central server, different users access web sites there. A commonly problem in web mining is the identification of user sessions. The sessions are further divided into transactions. Afterwards, the data has to be converted into a format that is readable by a mining algorithm. After the pattern discovery, several results are presented in [47, section 7]. The author suggests frequent itemsets and association rules for making successful advertising or improving the website structure. Frequent sequential patterns can reveal the navigation behavior of the users. This paper is related to this thesis, because it illustrates web usage mining in practice. Moreover, the click stream data is similar to the interaction stream. Thus, the work shows how different types of patterns can be mined from such a stream.

Another related paper *Web Usage Mining - Languages and Algorithms* [73] focuses more on the representation of the web log. While the web site structure is described in Extensible Graph Markup and Modeling Language (XGMML), the web log is expressed in Log Markup Language (LOGML). However, this paper uses also web usage mining algorithms to gather frequent sets, frequent sequences and frequent subtrees. Once a LOGML Generator [73, section 4] creates a web log, the Frequent Pattern Mining (FPM) framework discovers patterns [73, section 5]. A simple idea is to mine frequent page hits by users, however the author suggests also mining link sequences to receive frequent access paths. If only forward accesses are taken into account, frequent subtrees can be mined. Additionally, the author suggests mining frequent subgraphs by interpreting a web site with forward and backward references. [73, Section 5.3] shows the final results. Frequent sets are illustrated in [73, section 5.3.1]. More interesting frequent sequences are reported in [73, section 5.3.2]. The paper uses a maximal forward approach for transaction identification and the SPADE sequence mining algorithm for discovery. This related paper shows again that it's possible to use frequent pattern mining algorithms on a web log in a particular format.

Finally, *Automatic Personalization Based on Web Usage Mining* [56] focuses using web usage mining for anticipating user behavior and customization. The article works out in detail how to prepare the web data [56, Data Preperation] and how to use various data mining algorithms to discover the desired usage profiles [56, Discovery of usage profiles]. This is essentially not new and was covered sufficiently by the two previous works above. However, this article focuses on giving recommendation based on mined profiles [56, From profiles to recommendations]. The heart of this approach is a recommendation engine which calculates a so called *recommendation set*. This set contains objects (e.g. links, ads, text, products) and is computed by matching the current user profile against aggregate profiles. Three factors are suggested to determine the recommendation set: (1) A short-term history of a user, (2) aggregate usage profiles matching and (3) recommendation significance measurement. However, the author admits that "it is also possible to directly use patterns discovered as part of the association rule (or sequential pattern) discovery to provide recommendations" [56, From profiles to recommendations]. However, the author uses a n-dimensional URI vector for active session and profile. Hence, a distance or similarity measurement calculates suitable profiles with a matching score for recommendation. The end result is a web personalizer system [56, The WebPersonalizer System]. This related article shows that web usage mining can also be used for recommendation. Hence, in the GUI user assistance could be implemented by recommendation

in the same way. The discovered patterns serve as a user model for GUI navigation behavior. However, this is only an outlook in the future.

2.4.2 Graph Mining

On the other side, the related paper *Complete Mining of Frequent Patterns from Graphs: Mining Graph Data* [26] illustrates web browsing analysis based on graph mining. Ignoring the detailed explanations about the graph mining algorithm, [26, section 5.1] shows a specific application on web browsing. The authors assume an existing graph of hyper linked web sites. Additionally, they expect an access history of the user in form of sequences. Based on an existing access history, transactions are identified, where a pause of 5 minutes between two accesses indicate a transaction separation. The algorithm uses the graph-structured data to improve the frequent pattern result. In particular, it "can derive all frequent induced subgraphs from both directed and undirected graph-structured data having loops (including self-loops) with labeled or unlabeled nodes and links" [26, abstract]. The results reveal three aspects: (1) Some frequent patterns are valid traversals through the web site structure. (2) Other frequent patterns show cases where the user navigates from one site to another, however is forced to use a long path over other sites. (3) At last, there are frequent patterns that have more then one component (two or more subgraphs). The author suggests an additional link between these components such that a user has the possibility to navigate between these groups of URLs. The related work demonstrates how graph-structured data can help to find frequent patterns. Similarly, the GUI elements are also structured as a tree and moreover interactions with GUI elements can be interpreted as graphs. Thus, similar graph mining techniques can be applied to interaction graphs.

2.4.3 Process Mining

Two related paper in the context of process mining are presented below.

The first paper *Discovering Hierarchical Process Models Using ProM* [11] demonstrates the usage of the Process Mining Framework (ProM) to discover hierarchical process models. Hierarchical process models can help to deal with less-structured processes and fine-grained event logs. The hierarchy is obtained by determine new abstractions over old abstractions. The paper presents a two-phase approach [11, fig. 1]: (1) Find abstractions in the event log (level of granularity) and (2) discover the process maps. The approach is implemented with the ProM tool that consists of plugins. The Pattern Abstractions plugin is applied in the first phase [11, section 2]. It discovers common execution patterns (tandem arrays and maximal repeats). The patterns are measured with different metrics and thus can be filtered by thresholds. Patterns which are closely related form an abstraction. The abstractions can be labeled with meaningful names. Finally, the event log is transformed by replacing the patterns with their abstractions in separate sub-logs. The Fuzzy Miner plugin is applied in the second phase [11, section 3]. It creates process maps with a threshold, which defines the level of abstraction: An higher threshold relates to an higher abstraction. Abstractions are clusters of activities. The plugin uses the result of the previous plugin. Hence, the clusters correspond to abstractions found by the Pattern Abstractions plugin. The author suggests this approach "to create maps that (i) depict desired traits, (ii) eliminate irrelevant details, (iii) reduce complexity, and (iv) improve comprehensibility" [11, section 4]. This related work has a similar problem: "Events logs contain fine-grained events whereas stakeholders would like to view processes at a more coarse-grained level" [11, section 1]. This is also true for the acquired interaction log. Because every click and shortcut is recorded, the data consists of "fine-grained events". The discovered patters are intended to determine "a more coarse-grained level". Ideally, every pattern represents an intentional action.

The second paper *Process Mining Can Be Applied to Software Too!* [74] applies, as the title suggests, process mining in productive software systems to extract real software usage. The industrial paper shows two experience reports based on ticket reservation systems [74, section 2]. The first report is about an European touristic system which uses the touristic protocol *TOMA*. Based on the protocol they extract an event log, where every event has "an activity name, a timestamp, a user id, a booking code and a notification" [74, section 2.1.1]. The tool *Disco* mines process models based on a fuzzy mining algorithm and highlights frequent activities. They discovered negative behavior [74, section 2.1.2], positive behavior [74, section 2.1.3] as well as typical workflows [74, section 2.1.4]. The latter helps developers to understand how users actually work. The second more interesting report is about a web-based e-trading Russian traveling portal. Based on server logs they use the following attributes for process mining: session ID, activity, user and time stamp. The activity contains a detailed description about the web interface access, for example "WINDOW-LOAD" or "CONFIRM SUBMIT-CLICK" (altogether 50 distinct activities). A fuzzy and heuristic miner is used from the ProM framework. Based on the results [74, section 2.2.2] they found out that four situations are reasons for leaving the portal: (1) Payment method verification, (2) Booking confirmation, (3) fare condition acceptance and (4) insurance policy removal. These insights lead to defective software. As before, typical workflows [74, section 2.2.3] are analyzed. A heuristic net model sheds light on the workflow and shows that "more than a half of all the cases did not follow the normal scheme" [74, section 2.2.3]. The results are used to improve the system and interface design. This paper is related to this thesis, because [74, section 3] encourage the usage of process mining in software environment. In case of normal rich client applications the author suggests "to make logging on the client side" [74, section 3], like the interaction observer does.

However, event logging is assumed to be implemented "using listeners (observer pattern) for different GUI forms and widgets" [74, section 3]. This seems impractical and that's why the interaction observer is designed as an application software independent listener. Moreover, a similar structure of event logs are found: A user activity is the combination of an action with an object. Similarly, an event (definition 4) and an EOI (definition 10) represent an interaction.

2.4.4 Bayesian User Modeling

Finally, the last related paper *The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users* [46] manifests the vision of Desktop Usage Mining. The paper uses Bayesian user models to describe goals and needs of a user. The Bayesian user models [46, section 2] are basically Bayesian networks. Thus, it is possible to receive probability distributions by observed evidences. While *goals* are subtasks of the user, *needs* are actions or information that help to accomplish the goals. The needs forces the user to perform "sequences of user actions recorded as the user interacts with a mouse and keyboard (...)" [46, section 2]. Additionally, explicit queries in words are also considered. In the end, the main goal is to assist the user with autonomous actions. [46, section 3] studies assistance based on human experts and human subjects. The experts have to guess the subjects goal and give assistance, while only watch their interface, mouse and keyboard activity. They found out that experts can observe and identify user goals. However, misperceptions of goals distract the user. These insights were used to manually construct Bayesian networks for diagnosing goals [46, section 3.2]. The monitoring becomes a temporal reasoning problem, because "observations seen at increasingly earlier times in the past have decreasing relevance to the current goals of the user" [46, section 4]. To receive a stream of expressive user actions the application software Excel is altered [46, section 5]. These atomic events are transformed to higher level observations base on temporal pattern recognition. Together with a Bayesian user model, containing approximately 40 problems in Excel, the Lumière/Excel system is complete [46, section 6]. Given a stream of user actions the system displays an inferred probability distribution of needs [46, section 6.4]. The needs are recommended online help topics of Excel. This paper is related because the presented system "gaining access to a stream of events from software applications" [46, abstract], similar to the interaction observer. They build manually an application software and problems dependent Bayesian user model for inference. However, this thesis focuses on an application software independent view. The higher level observations in the paper can be compared with the desired patterns. As an outlook, patterns could be used to assist the user in a similar way the paper demonstrated.

2.5 Approach

Initially, this thesis tries out exploratively several approaches discussed in the background section. These selected strategies (abbrev. S) are *sequential pattern mining* (S1), *graph mining* (S2) and *process mining* (S3). This was done to get a feeling what is out of the box possible with current approaches and technologies. The trial and error was important to realize disadvantages as well as advantages of some approaches in other disciplines. Besides that, this thesis comes up with an own approach based on *n-grams* (S4). This strategy is designed to overcome the issues of the former three strategies.

However, the following sections are structured as follows: First of all, **reference patterns** are annotated by selected participants. Afterwards, the interaction log is **preprocessed** on the basis of the reference patterns. Finally, pattern discovery, implemented by four strategies, mines patterns on the preprocessed interaction data. **Sequential pattern mining** (S1) mines frequent sequences from a sequence database. **Graph mining** (S2) discovers frequent subgraphs from a given graph database. **Process mining** (S3) interprets the interactions as events and mines patterns from a given event log. The **n-gram based** approach (S4) finds frequently reappearing n-grams.

Beyond, the **evaluation** section analyses the preprocessing and the mined patterns.

2.5.1 Reference Patterns

It is crucial to know what is the outcome of the mining process. To get a better idea how patterns look like, the interaction log has to be reviewed by humans. These persons provide information about the appearance of patterns. We call these patterns *reference patterns*.

Reference patterns are patterns which are annotated by experts. These experts are three selected participants who annotated their own interaction log. With the help of a Pattern Annotation Tool the annotators had the ability to select interactions and mark them as a pattern.

The following instructions were given: Initially, an annotator has to load his file that contains the complete interaction log. After loading is completed an interaction table shows the following attributes while each row represents an interaction:

Id The unique ID of the interaction. A sequence uses the interaction ID to refer to the related interaction.
Time Stamp The moment the interaction was recorded.
Event The event that initiated the interaction.

EOI (Element of Interest) The element which the user interacted with. A selected row reveals detailed information of the EOI.
GEOI (Generalized Element of Interest) A generalized element which the user interacted with (see section 2.5.2).
Crawl The hash of the crawl (set of elements).
User The user who is always the same because each file is user-specific.
Program The application software which the user interacted with.

Two lists has to be filled by the annotator: (1) The pattern class list is a catalog of pattern classes. A pattern class has a name and holds a set of pattern instances. (2) The pattern instance list is a container for pattern instances. A pattern instance is a sequence of interactions.

The task was to reproduce the made interactions and to find sequences which deserve names. The annotator has to ask herself/himself why (s)he performs the given sequences of interactions. If an answer was found the annotator had to name the intentional action. Thus, a pattern class has to be created. The sequence of interactions that represents the action has to be added as a pattern instance. Frequent performed actions curse different pattern instances.

In a manual post-processing step the annotations were reviewed and prepared in the following way: An interaction pattern is described by a name, a context, a problem and a solution. The name identifies the pattern and is used for communication. It is only valid in the context because a different context can have the same pattern name. The context refers to the application software where the pattern was discovered. The problem describes the intentional action of the user who wants to accomplish a specific task. The solution reveals in a regex-like syntax (regular expression) what has to be clicked and/or pressed to accomplish the specific task in the GUI. Sometimes more then one solution is possible to solve the problem. Some application softwares allow different click paths to accomplish the same pattern.

The following patterns were recognized by the annotators:

Name GIT PULL
Context SourceTree 1.6.4.0
Problem The user wants to incorporates changes from a remote repository into the current branch [21].
Solution (1) Mouse Left Click Button-49483 Pull, Mouse Left Click Button-64819 Ok
(2) Mouse Left Click Button-49483 Pull, Mouse Left Click ComboBox-82855, ListItem-83030 master, Mouse Left Click Button-64819 Ok

Name GIT FETCH
Context SourceTree 1.6.4.0
Problem The user wants to branch and/or tags from one or more other repositories [20].
Solution Mouse Left Click Button-49482 Anfordern, Mouse Left Click Button-82419 Ok

Name GIT COMMIT
Context SourceTree 1.6.4.0
Problem The user wants to store the current contents of the index in a new commit along with a log message from the user describing the changes [19].
Solution Mouse Left Click TreeItem-66833 Arbeitskopie, (Mouse Left Click TabItem-66208 Dateistatus)$^{+}$, Keyboard Control + RETURN Edit-66223

Name DISCARD FILE CHANGES
Context SourceTree 1.6.4.0
Problem The user wants to discard his file changes.
Solution (Mouse Left Click TabItem-66208 Dateistatus)$^{+}$, Mouse Left Click Button-67057 Block verwerfen, Mouse Left Click Button-69139 _OK

Name SHOW DATA TABLE
Context pgAdmin III - PostgreSQL Tools 1.18.1
Problem The user wants to look at rows of a data table.
Solution (1) Mouse Right Click TreeItem-*, Mouse Left Click MenuItem-51351 Daten anzeigen, Mouse Left Click MenuItem-51360 Die letzten (100) Zeilen zeigen
(2) Mouse Right Click TreeItem-*, Mouse Left Click MenuItem-51351 Daten anzeigen, Mouse Left Click MenuItem-51360 Die oberen (100) Zeilen zeigen

Name Send Email
Context Microsoft Outlook 14.0.7113.5000
Problem The user wants to send an email.
Solution (1) Mouse Left Click Pane-104289 Nachricht, Mouse Left Click Button-39269 Antworten, (...)*, Mouse Left Click Button-104282 Senden
(2) Mouse Left Click Pane-104289 Nachricht, Mouse Left Click Button-39270 Allen antworten, (...)*, Mouse Left Click Button-104282 Senden
(3) Mouse Left Click Button-39279 Neue E-Mail-Nachricht, (...)*, Mouse Left Click Button-104282 Senden

Name Delete Email
Context Thunderbird 24.6.0
Problem The user wants to delete an email.
Solution Mouse Right Click DataItem-*, Mouse Left Click MenuItem-4249 Löschen

Name Empty Trash
Context Thunderbird 24.6.0
Problem The user wants to delete an email.
Solution Mouse Right Click TreeItem-2921 Papierkorb, Mouse Left Click MenuItem-90856 Papierkorb leeren, Mouse Left Click Button-90866 Ja

Name Read Email
Context Thunderbird 24.6.0
Problem The user wants to read an email and close it afterwards.
Solution Mouse Left DoubleClick DataItem-*, (...)*, Mouse Left Click Button-2988

Name Retrieve Email
Context Thunderbird 24.6.0
Problem The user wants to retrieve new emails manually.
Solution (1) Mouse Left Click Button-2965 Abrufen
(2) Mouse Left Click Button-2950 Abrufen Neue Nachrichten empfangen, Mouse Left Click MenuItem-88972 Alle Konten abrufen

Name Print Document
Context Microsoft Office Word 2013 15.0.4641.1000
Problem The user wants to print a document.
Solution Keyboard Control + P Edit-30881 Kopfzeile -Abschnitt 1-, Mouse Left Click ComboBox-31097 Welcher Drucker, Mouse Left Click Menu-31118, Mouse Left Click Button-31071 Drucken Button-31071 Drucken

Investigations of the reference patterns give new insights below.

The patterns reveal that they consists of many MenuItem and Button controls. These elements represent a functional aspect of the pattern. Besides that, keyboard shortcuts seem also have a functional matter. A shortcut can begin (see Print Document) or end (see Send Email) a pattern. Other elements carry an informative aspect for the user. Some alterable elements are used to setup parameters. This gives the possibility to classify interactions based on the control type of the EOI.

The most patterns are short and consists of approximately three interactions. However, complex patterns can occur, too. These patterns have a beginning and an end. The problem is that various interactions happens in between. This is denoted with a (...)* in the Send Email and Read Email pattern. On the other side, the pattern length depends on the task. The Discard File Changes serves as an example. Depending on the discarded file changes, the pattern grows arbitrary. This is denoted with "(Mouse Left Click TabItem-66208 Dateistatus)$^{+}$" and means one or more mouse clicks on a tab item called file status.

Some reference patterns have two or more solutions. The application software provides many ways to perform an intentional action. In the most cases, menu items or buttons are substituted by shortcuts. Other solutions express a slightly different intention. The Send Email pattern makes this clear. In contrast, the Git Pull pattern shows two

solutions which differ in expressiveness. Solution (1) represents a common task, while solution (2) allows additionally certain parameters.

In some cases, a regex-like syntax has to be used. The reason is that different EOI instances are possible. The SHOW DATA TABLE pattern (as well as the DELETE EMAIL pattern) illustrates this fact. The "TreeItem-*" annotation expresses that different tree items are used with the same functionality. A generalization would help to match different tree items to its tree parent. The particular tree item is for the pattern irrelevant.

These considerations help to preprocess the data and rediscover the reference patterns and other patterns.

2.5.2 Preprocessing

Before algorithms can discover patterns the raw interaction stream has to be preprocessed. This preprocessing is organized as follows: First, only one-to-one relationships between user and application software are considered. Second, the EOI and its generalized form is determined. Third, interactions are classified to seven classes based on the control type of the generalized EOI. Fourth, repeating EOIs are removed to clean the interaction data. Finally, transactions resp. sequences have to be identified for approaches which require a transaction database. Two supporting and four main transaction identification approaches are investigated.

The complete interaction stream includes all observed users with all their used programs. That's why the stream has to be divide into appropriate partitions. The full stream will be divided depending on the focused relationship between user and application software. There are three different types of relationships where *n* users interact with *m* application softwares, denoted with *n*:*m* (*n* to *m*): One user interacts with one application software (1:1), one user interacts with *m* application softwares (1:m) and *n* users interact with one application software (n:1).

(1:1) This relationship is the simplest and most suitable for discovering patterns. A stream of interactions is extracted depending on a selected user and his/her used application software. The focus lies in patterns which occur between one specific user working with one specific application software. Patterns depend on users and how they use application softwares. Every user evaluates his own discovered patterns. The restriction helps to compare the results. This thesis will exclusively investigate 1:1 relationships.

(1:m) This relationship shows how the user interacts with different application softwares. This observation can reveal patterns which show up correlations between application softwares. Sometimes a task needs various application softwares and the user has to switch between them to accomplish this task. For example, using a presentation application software and using a diagram application software is very likely.

(n:1) This relationship shows commonly used elements and click paths of the application software by different users. However, various users that use the same application software often have different versions. Thus, these versions have to be aligned.

For every interaction an EOI has to be chosen because the raw interaction (see definition 1) doesn't point to exactly one element. That's because some uncertainty is modeled in the database. In case of a keyboard event the focused element could be determined or not (null value). In case of a mouse event three methods, which can also fail, are used to determine the element under the cursor. In section 1.5.4 is explained in detail how the final EOI is determined.

If the EOI is always determined correctly, every mouse click and every shortcut refer to an element. It has been shown that these elements provide too specific information. For example, the approach can determine the exact clicked data item, tree item or list item. The reference patterns have shown that this specific knowledge is not important. If every data (tree or list) item element would be distinguished, patterns that should be determined equal would be unequal. Even worse, because some buttons have an image on top, the EOI of a click on those button is an image element. Usually, this information is too specific. To solve these problems the EOI has to be *generalized*.

The generalization algorithm uses the control type [70] of the elements to generalize them. The algorithm is a rule based approach and divides in three parts: (1) For some elements the generalization is not necessary, (2) other elements could be generalized to more suitable elements and (3) item elements could be generalized to their container element (parent).

(1) There is no generalization necessary for the following control types:

- MenuItem
- Button
- Tab
- Tree
- Table
- Document
- ToolBar
- Group
- Window
- Edit

(2) For the control types below a generalization can bring a benefit. Text and Image are listed because sometimes buttons have text or images on them.

- Custom
- Separator
- Text
- Image
- Hyperlink

The algorithm tries to find a more suitable element in the ancestors of the element that has to be generalized. If an ancestor has one of the following control type, the ancestor is the generalized element.

- MenuItem
- Button
- Tab
- Tree
- Table

(3) The following control types are items that are contained in specific containers:

- TreeItem is contained in the container Tree
- DataItem is contained in the container Table
- TabItem is contained in the container Tab

The algorithm tries to find the container in the ancestors of the element that has to be generalize. Fortunately, the GUI tree can be used to find the ancestor easily.

Each interaction can be assigned by the control type of the generalized EOI to one of the six classes below. There are 38 control types [62] specified in Windows 7 that can be distributed to these classes.

Structural (S) These elements structure other elements, but don't visualize information nor call functions. They are containers and exist because humans orient themselves with a layout. Such elements provide layout, structure and semantic correlation.

The following control types assign elements to this class:

- Menu
- Window
- Group
- Pane
- TitleBar
- List
- DataGrid
- Header
- MenuBar
- Tab
- Table
- ToolBar
- Tree

Semi-Structural-Informative (SI) These elements structure the GUI but are at the same time no containers. There is only one control type that fits to that definition: the Separator. The separator is used in menus and menu bars to separate the containing elements. The presence of the separator clusters elements in a meaningful way. However, the separator is no container for elements.

Informative (I) These elements visualize information. They are only informative, but not alterable.

The following control types assign elements to this class:

- TabItem
- TreeItem
- DataItem
- ListItem
- Text
- Hyperlink
- Image
- HeaderItem
- ProgressBar
- StatusBar
- Thumb
- ToolTip

Semi-Informative-Functional (IF) These elements visualize information but at the same time manipulate information, because they are alterable.

The following control types assign elements to this class:

- Slider
- Spinner
- RadioButton
- CheckBox
- Calendar
- Edit
- Document
- ScrollBar
- ComboBox

It should be noted that the combobox is both informative, functional and also structural, because it is a container for listitems. For convenience, we assign the combobox to this class.

Functional (F) These elements are used to invoke a function (or subroutine) which manipulates something. They visualize no information nor structure elements.

The following control types assign elements to this class:

- Button
- MenuItem

It should be noted that only menu items, which are leafs in the tree, are functional. The other menu items open new menus.

Semi-Functional-Structural (FS) These elements invoke a function but at the same time structure elements. There is only one control type that fits to that definition: the SplitButton. The splitbutton is a button but has additional buttons if the default button is not the appropriate one. This element structures semantically equal buttons and is simultaneously a button.

The control type Custom is left over. Because a custom element could be everything, we can't assign a class from above to it. That's why a seventh class *None* exists. In addition, interactions which have no generalized EOI are assigned to this class, too. These interactions are initiated by an observer or process event and don't have an EOI.

It is fair to acknowledge that in the GUI arbitrary behavior can be programmed. Thus, programmers have the freedom to use control types in a non-standard way. For example, a checkbox can behave like a button and vise versa. However, this classification will help to focus on standard functional elements.

A particular case are keyboard events. The usage of a shortcut is also a functional interaction. Especially, menu items can sometimes be accessed with assigned shortcuts. Thus, the accomplishment of a shortcut is technically the same as the click on the related menu item. In another case users perform shortcuts on focused text boxes (Edit control type) and documents (Document control type). Such shortcuts (for example copy, Ctrl+C) focus on alterable elements but are in fact invoked subroutines. That's why in both cases the control type of the focused element doesn't reveal the classification of the interaction. Hence, keyboard shortcut are always assigned to the functional class, regardless of the focused element's control type. In particular, a keyboard event is a shortcut if at least one of the modifiers (alt, control or shift) pressed. However, keystrokes which are not shortcuts are classified as *None*.

Interactions are removed that again interact with the same EOI. Because users are not careful with their clicks and shortcuts, the interaction stream consists of repeating interactions with the same element. These interactions don't give a benefit for the strategies and cause patterns with repeating interactions. That's why they are removed to clean the interaction stream.

The algorithm focuses on the ID of the EOI. If consecutive interactions point to the same element, the first interaction will be kept, while the remaining interactions will be removed. As an example, for the given sequence $\alpha = (1,1,2,2,3,3,3,2,1,1)$ the algorithm removes the repeating elements $(1,\mathbf{1},2,\mathbf{2},3,\mathbf{3},\mathbf{3},2,1,\mathbf{1})$ and returns $\beta = (1,2,3,2,1)$.

Finally, this thesis will introduce two supporting and four main transaction identification (abbrev. TI) approaches. Transactions are meaningful clusters found in the log data. Because the log is a long sequence of interactions, transactions can be seen as meaningful sequences. An interaction sequence is meaningful, if it represents an intentional action. Strategy 1–3 expect a sequence database as input (strategy 2 expects a graph database, however every sequence is transformed into a graph). A sequence database is a set of sequences. The output of the strategies depends on the sequence database input. For example, frequent sequential patterns (S1) are sequences that occur frequently in such a sequence database. Therefore, approaches are necessary to identify transactions resp. sequences in the interaction stream to form this sequence database.

The following two supporting approaches are used to find obvious transactions which however are still too large.

The first trivial approach is to split the stream at every interaction that has an observer or process event. Fortunately, the observer events reveal the beginning and end of a working session with the PC. In contrast, the process events point out context switches. Process close events determine an end of a working session with an application software. Therefore, the assumption was that no pattern consists of interactions with observer or process events. Two interactions were separated if the left or right interaction has such an event. This approach returns obvious transactions that are however still too coarse-grained.

Another fairly trivial approach is to split the stream where two consecutive interactions are remarkable temporally separated. This idea comes from the fact that a pattern is limited in time. A constant time span decides if two consecutive interactions are separated. This method detects obvious "time holes" between interactions. It returns transactions of various length, depending on the time span parameter. A long time span is used to discover obvious transactions. However, this results in a still too coarse-grained fragmentation.

The following four main transaction identification approaches try to find meaningful transactions.

(TI1) A different adequately trivial approach is to split the stream where a functional interaction is made. In particular, this includes mouse interactions with either a button or a menu item as well as keyboard interactions which are shortcuts. The assumption is that buttons or menu items indicate the end of a task resp. pattern. In the same way shortcuts can indicate complete tasks. Analyses on reference patterns showed that the patterns end always with some functional

interactions. For example, in case of a mouse interaction the click on an OK button confirms a task. Another example, in case of a keyboard interaction the shortcut *Ctrl + Return* sends an email.

The following three transaction identification approaches are widely used by web usage miners who want to identify meaningful clusters in a raw server log.

(TI2) [27, Section 3.2.1] explains transaction identification based on a reference length. This approach assumes that pages can be classified as either a *navigation* or a *content* page. While navigation pages are used to navigate to a desired page, content pages have the desired information. The assumption is that a user spends less time on navigation pages and in contrast more time on content pages. The cut-off time, which decides the class membership, can be calculated with a guess of the percentage of navigation pages in the log. In particular, [49, p. VI.] shows the calculation of the cut-off time in detail. In this case, web pages can be reinterpreted as GUI elements. In doing so, an access to a web page is similar to an interaction with a GUI element. Hence, two consecutive interactions will be separated if the time span between them is longer then the calculated cut-off time. This approach requires a parameter. The parameter comes from the fact that the reference length is usually an exponential distribution. While content pages make up the upper tail, navigation pages make up the lower end. Different parameter values are investigated in the evaluation section. However, in section 1.7.1 an example participant is illustrated which demonstrates the interaction time histogram 8. It is similar to a moved exponential distribution with a negative exponent. That's why the approach could also be used in the GUI domain. However, it is questionable if the GUI also distinguishes between navigation elements and content elements. Moreover, it is dubious if content elements determine an accomplished task.

(TI3) [27, Section 3.2.2] describes the maximal forward reference approach. This approach actually originates from the paper *Data Mining for Path Traversal Patterns in a Web Environment* [17]. It analyses the backward references in the log. A backward reference is a reference which occurred earlier in the reference history. In contrast, a forward reference is not contained in the history yet. The algorithm keeps track of the references with a history. Whenever a reference occurs, which is already contained in the history, a backward reference is determined. In this case, the history is cleared and filled with the already seen reference. For example, the sequence $\alpha = (a, b, c, a, c, d, e, c, f)$ is separated in $TI3 = \{(a, b, c), (a, c, d, e), (c, f)\}$. In contrast to the paper, the forward references are not concatenated with the previous references [27, Section 3.2.2, example]. No parameter is necessary for the detection. However, this thesis uses two separate clues to detect a backward reference: (1) the crawl and (2) the generalized EOI. (1) The crawl ID represents a state of the application software. Whenever a user reaches the same state of the application software, a backward reference is determined. Similarly, web pages contain HTML elements, crawls contain GUI elements. That's why a crawl seems to be an appropriate indication. This assumes that patterns never contain interactions with same application software states. (2) The generalized EOI ID is more detailed then the crawl ID. Whenever a user interacts with the same element, a backward reference is determined. This assumes that patterns never contain interactions with the same element. The approach makes sense, however not every accomplished task will end in an already visited state of the application software (crawl) or EOI.

(TI4) [27, Section 3.2.3] expounds the transaction identification based on a time window. This time window ensures that references occur in a specified time interval. An average interaction duration associates a meaningful transaction. For example, assume that the following sequence of numbers are time stamps in seconds: $\alpha = (1, 5, 8, 10, 15, 33, 45, 46, 50)$. If we define a time window of 10 seconds, the following transaction set would be calculated: $TI4 = \{(1, 5, 8, 10), (15), (33), (45, 46, 50)\}$. The algorithm put transactions together, where begin and end have a time difference not greater than the given time window. The necessary parameter is a fixed time span defining the time window. The evaluation investigated different values. Interaction time is different among users and tasks. That's why it is questionable if the transactions represent accomplished tasks.

In a post processing step two filters are used on the set of transactions. First, only transactions are considered which have a minimal length. This parameter helps to remove small transactions which confuse the algorithms rather then give a benefit. Second, only transactions are considered which contain at least one functional interaction. This increases the chance that the algorithms return again patterns with some functional interactions. Additionally, interactions without a generalized EOI are removed. That's because all strategies below use the generalization of the EOI. This filter removes automatically interactions containing observer and process events as well as erroneous interactions.

2.5.3 Strategy 1: Sequential Pattern Mining

The first strategy is the usage of a sequential pattern mining algorithm on the preprocessed interaction stream. Click-stream data is commonly analyzed with sequential pattern mining. Because interactions occur in sequence, sequential patterns are suitable for the problem.

The approach applies a particular algorithm which is presented in the paper *VMSP: Efficient Vertical Mining of Maximal Sequential Patterns* [34]. The paper introduces frequent sequential patterns. These patterns have a minimal frequency above a specified threshold called *minsup*. However, the mining returns too many frequent sequential patterns. The frequent sequences are very similar and don't give a benefit for the user. That's why closed sequential patterns are investigated. "A closed sequential pattern is a sequential pattern that is not strictly included in another pattern having

the same frequency" [34, section 1]. The mining results in fewer closed patterns, however the outcome is still too large. That's why maximal sequential patterns are proposed by the paper. "A maximal sequential pattern is a closed pattern that is not strictly included in another closed pattern" [34, section 1]. This returns a very small result. The maximality eliminates very similar patterns and focuses on long patterns. This is useful for mining interaction patterns.

[34, Section 2] defines the problem of mining maximal sequential patterns. Practically every sequential pattern mining algorithm expects a sequence database as input. A sequence database is a list of sequences, while every sequence is an ordered list of itemsets. An itemset is an unordered set of items.

The raw interaction stream has to be transformed to a sequence database. That's why the stream is first preprocessed (see previous section). Four transaction identification approaches are applied to separate the stream in smaller sequences in four different ways. Each sequence in the resulting sequence database is a transaction. Every itemset of the sequence is a set containing one item. The single item is the generalized EOI ID. This approach ignores different events (mouse or keyboard) and focuses only on generalized EOI IDs.

Vertical mining of Maximal Sequential Patterns (VMSP) [34, Section 3] is a novel algorithm proposed by the paper. The search procedure [34, section 3.1] searches recursively for patterns. However, the three strategies enables the efficient search for maximal patterns.

Strategy 1 efficiently filters non-maximal patterns. In doing so, a structure stores all maximal patterns which are found. This structure is updated in two ways: Super-pattern checking adds only maximal patterns. Sub-pattern checking removes patterns from the structure, if they are not maximal anymore. This structure contains at the end all maximal patterns. For efficiency, three optimizations are implemented: Size check optimization, sum of items optimization and support check optimization.

Strategy 2 avoids super-pattern checks on frequent pattern generated by the recursive search procedure call. An obvious prefix is the reason for the non-maximality.

Strategy 3 prunes the search space. The co-occurrence information is used to build a map. Every entry maps to a set of succeeding items. The lookup reveals infrequent pattern joins in the generation process. Thus, a recursive call can be omitted and the search space is pruned.

The algorithm expects three parameters: (1) The minimal support threshold *minsup*. The support value of a pattern is the number of sequences of the sequence database where the pattern is contained. The minsup expresses the relative frequency of containments. A value of 0% would return every pattern, while a value of 100% would return patterns which are contained in every sequence of the database. (2) A minimal length of the patterns. This value can be used to remove trivial patterns with length one (see definition 21). (3) A maximal length of the patterns. However, long interaction patterns are desired. That's why this parameter is not used and set to an obvious high value.

The mining algorithm returns a list of patterns which fulfill the specified constrains. Every pattern is measured by the support value. Because every item is still a generalized EOI ID, the item is resolved to its corresponding GUI element. Finally, the GUI element patterns are descending ordered by the support value.

2.5.4 Strategy 2: Graph Mining

Another strategy is the usage of graph mining. The approach relies on the algorithm presented in the paper *gSpan: Graph-Based Substructure Pattern Mining* [86]. The algorithm "graph-based Substructure pattern mining (gSpan)" solves the problem of discovering frequent subgraphs. Frequent subgraph mining is similar to frequent sequential pattern mining. A graph database is a set of graphs. The approach finds all frequent connected subgraphs in the graph dataset. A subgraph is frequent if it occurs more then or equal to a minimal support value in the dataset.

However, the given data is an interaction log, a sequence of interactions. That's why a transformation is necessary to create a graph database. Two types of graphs are considered, while the latter is used:

(1) Every interaction stores the state of the application software as a crawl. The crawl has a unique ID. A crawl graph is a connected directed structure where the vertices are crawl IDs. An edge expresses a state change of the application software. Additionally, the edge contains the EOI which causes the state change. The mining results are subgraphs expressing frequent state changes. However, these patterns are not expressive enough. Indeed, commonly state changes could reveal interesting information, but this thesis focuses on element interaction patterns.

(2) Interactions point to the generalized EOI. Every generalized EOI has a unique ID. An element graph is a connected directed structure where the vertices are generalized EOI IDs. An edge expresses a consecutive interaction with the elements. The mining results are subgraphs expressing interaction sequences. However, keyboard interactions are not modeled accurately. Only the generalized EOI is considered without an related event.

Every transaction, identified by an transaction identification approach, is transformed to an element graph. Two consecutive interactions form an edge if both have an unequal generalized EOI. This comes from the fact that the gSpan algorithm ignores self-loops. Only graphs with at least one edge are considered and added to the graph database. This encourages more complex graph patterns. The transformation to graph structures allow modeling branches and loops.

The algorithm expects a graph database and three parameters: (1) The minimal frequency. This parameter prunes subgraphs which are not frequent enough. (2) The minimal vertex count. This value can be used to disallow small

graphs. Trivial graphs with one vertex could be filtered. (2) The minimal edge count. This value can also be used to reject small graphs. A higher value removes less complex graphs.

The mining algorithm returns a list of subgraphs which fulfill the specified constrains. However, a lot of similar patterns are found. Some patterns differ only in an additional vertex. That's why a similar maximal pattern discovery approach is applied (see previous section). In contrast, the approach is deployed *after* the algorithm returns the results. Maximal subgraph patterns are found by using a containment check. This check is applied on every pattern pair (g_1, g_2). If the vertex-set of g_1 is a superset of the vertex-set of g_2 and the frequency of g_1 is greater or equal to the frequency of g_2, g_2 is non-maximal. This returns only maximal frequent subgraph patterns.

In a postprocessing step, the vertex is resolved to its corresponding GUI element, because every vertex is still a generalized EOI ID. The algorithm returns a frequency value for each subgraph. Thus, the GUI element subgraphs are descending ordered by their frequency values.

2.5.5 Strategy 3: Process Mining

The third strategy is the usage of process mining. This approach usually extracts process models from a given event log. Unfortunately, real event logs are less structured and end up in spaghetti-like process models. The background research covered already that approaches exist to overcome these challenges.

One way is the usage of abstractions [10]. We investigated that the thesis' pattern definition is similar to the definition for sub-processes as conserved regions (repeats): "similar regions (sequence of activities) common within a trace and/or across a set of traces in an event log signifies some set of common functionality accessed by the process" [10, section 3.2]. That's why this approach applies the discovery of repeats. Three types of repeats are defined: maximal repeats, super maximal repeats and near super maximal repeats. Unfortunately, only the discovery of maximal repeats are implemented. Thus, in a first step maximal repeats are discovered. In a second step the approach discovers abstractions of patterns [10, section 4]. A repeat alphabet is a set of activities occurring in the repeat. Only the repeat alphabets of the maximal repeats are considered for the abstraction. A partial ordering (subsumption is the cover relation) on the repeat alphabets reveals maximal elements. These maximal elements are considered as abstractions of processes. This thesis interprets them as patterns (definition 21). Only maximal elements with two or more activities are considered to be non-trivial patterns. In a postprocessing step, a containment filtering similar to the graph mining strategy is applied, because still some duplications and subsets exist. The disadvantages are that an abstracted pattern becomes a *set* of activities, rather then a sequence. Because the temporal ordering is no longer available, they are actually honestly no patterns anymore. Moreover, no measurement is applied that values abstractions. However, the abstraction promises the identification of sub-processes or common functionality.

However, before this discovery can happen, the interaction log has to be transformed into an event log. "To be able to apply process mining techniques it is essential to extract event logs from data sources" [1]. The interaction log is such a data source, however still in an undesired format for process mining. An event log consists of cases (process instances, activities or events) which are executed in a system [84, section 1.2]. By recording events it expresses what actually happens. A trace is a sequence of events which describes usually a completed interaction with the system. This can be considered as a transaction.

An event log can be represented in different formats. This thesis uses the Extensible Event Stream (XES) format which is recommended by the IEEE Task Force on Process Mining [80]. XES "is an XML-based standard for event logs" [83]. An XES log [36] consists of traces, which again consists of events. However, they are only give structure to the event log. Attributes [36, section 2.2] can be applied to log, trace and event objects for storing specific information. An attribute is a key-value pair and has a type. The type can be a string, date, int, float, boolean or ID. Moreover, extensions [36, section 2.6] specify the semantic of special attributes. They "introduce a set of commonly understood attributes (...)" [36, section 2.6]. [36, Section 4] lists some standard extensions which define commonly used attributes. The concept extension [36, section 4.1] is important for storing a name in logs, traces and events. The `concept:name` attribute distinguish the objects by given names. However, equal named objects are considered to be equal. The lifecycle extension [36, section 4.2] lets events specify a lifecycle transition from a transactional model. A standard transactional model is given. The `lifecycle:transition` attribute can be set to one of 13 transitions. The time extension [36, section 4.4] defines a time stamp attribute for every event. Thus, time information can be stored which is necessary for many analysis techniques. The `time:timestamp` attribute captures the time and date when an event occurred.

The log of interactions (see definition 1) is converted to the structure and attributes of the event log. Every user-program pair defines a log. The log's name is the concatenation of the participant name and the application software. Traces are transactions which are identified by the transaction identification approaches (TI). The concept name of a trace is the time stamp of the first and the last event, separated by a "-" symbol. The transactions don't have a common beginning and ending interaction. Usually, in this case artificial interactions are added. They express the beginning and ending of a task. In a similar way to the *Add Artificial Events* plugin [18], two events are added: (1) At every beginning of a trace an event is inserted. This event is called "Artificial Task Start" (concept name) and its time stamp

is one second before the first event's time stamp. (2) At every ending of a trace an event is inserted. This event is called "Artificial Task Stop" (concept name) and its time stamp is one second after the last event's time stamp. An event represents an interaction. The concept name of an event consists of two parts: (1) The interaction's event and (2) the interaction's generalized EOI. (1) Only mouse or keyboard events can occur. In the case of a mouse event, the representation contains "Mouse $entity\ style$" (see definition 8). For example, this could be "Mouse Left Click". In the case of a keyboard event, the representation contains "Keyboard $modifiername + key$" (see definition 7). The modifier name is one of "Alt", "Control" or "Shift", depending on the pressed modifier. An example could be "Keyboard Control + V". (2) The generalized EOI is represented by a concatenation of its ID and identification properties (see definition 11). This representation contains "$controltype\text{-}id\ name\ description$". It's not unusual that the name and description properties are empty. Examples are "Tab-179701 Console", "Edit-179828" and "Button-191045 OK". The interaction's time stamp is the event's time stamp. Finally, every event has the transition "complete" from the standard transactional model. This indicates that every interaction execution is complete.

2.5.6 Strategy 4: N-Gram Based

The transaction identification has taught that it's hard to know when a task begins and ends. Reference patterns showed that they are short sequences which reappear in a slightly different way. Because of these insights the patterns are discovered with an n-gram based approach.

An n-gram is a sequence with n contiguous items from a given sequence. They are commonly used in natural language processing. Usually, n-gram models are used for predicting a word from earlier seen words [13]. However, this thesis uses interaction n-grams to find frequently reoccurring and slightly different sequences. The adjustable n can be used to generate arbitrary long sequences. Thus, the uncertainty of the pattern length is taken into consideration.

Moreover, the thesis uses skip-grams which are a generalization of n-grams [37]. Skip-grams allow that k items can be skipped in between. [37, Section 2] gives a definition of skip-grams: The paper allows k or less skips in the k-skip-n-grams. However, this thesis will allow only exactly k skips. The skips help to model the fact that patterns can reappear in a slightly different way. Moreover, more complex patterns can be discovered that are not a static click path. They have many variations and branchings, however represent the same intentional action. In allowing interaction holes these patterns could be found too.

The previous strategies and the reference patterns above have taught that looking for frequent skip-grams is not enough. High frequent skip-grams reveal a lot of irrelevant interaction sequences. These sequences contain mainly interactions with non-functional elements. That's because users primarily click on informative elements that show new content while never using any functionality. To solve this issue the approach will consider exclusively skip-grams which contain at least one functional interaction. The classification explained previously helps to detect these skip-grams. After this filtering only *functional* skip-grams are left over.

For further cleaning, skip-grams which contain interactions initiated by observer or process events are removed. Observer events only give insights when the observation begins or ends. Patterns that contain observer events are interrupted interaction sequences. Because of the observation error they can not take for granted. Process events only clarify the opening and closing of a process. This thesis focuses exclusively on a 1:1 relationship between the user and the application software. Hence, resulting patterns containing application software switches are irrelevant.

After the filtering and cleaning process, a set of functional skip-grams is left over. Reference patterns showed that they reappear in the interaction log. That's why the approach searches for equal functional skip-grams. However, patterns can occur in a slightly different way. The reason is that functionality can be used on different inputs. Interactions around a functional interaction variate, however the functional meaning stays the same. For example, different tree items are clicked, but afterwards the same functional element is used. In that respect, the same intentional action is expressed in various instances.

Thus, the approach has to define whenever a skip-gram is equal to another. The skip-grams are compared item-wise, which is always possible because the n and k is fixed. Interactions, that are initiated with a keyboard event, are compared with the key value. In contrast, strategies 1 and 2 never modeled explicit the related event. Thus, keyboard events are not considered enough. Interactions, which are initiated with a mouse event, are compared with the generalized EOI. The generalization helps to overcome the problem of various instances which express the same intentional action. Functional interactions are not affected by the generalization, because the Button and MenuItem control type isn't generalized. A functional interaction is equal to another, if the IDs of the (generalized) EOI are equal. However, non-functional interactions are compared by their classification. Is the classification equal, the control type of the EOI is used for the comparison. An equal control type defines that non-functional interactions as equal. The reason is that this tolerant comparison discovers syntactic unequal but however semantic equal skip-grams.

After receiving a list of equal skip-grams, some of them are variations of the same functional interaction. This is manifested by the fact that the same functional interaction occurs in various positions in the skip-grams. Besides that, the elements around the functional interaction can change heavily. Hence, functional interactions can occur in different contexts.

Some functional interactions are task independent. The interactions before and after these functional interactions have a high variability. It seems that there is no fixed usage of the related function. For example, such task independent buttons are the minimize, maximize and close button of a window. Another example would be a button which closes a viewed document (e.g. an email). This button is also used in many contexts because before or after the document is closed various task are made.

In contrast, some functional interactions are task dependent. These interactions correlate with certain interactions which occur before or after. An example would be a "save" button. Every time this button is used a "filename" text box is interacted with before. There are no other possibilities what could be happen before this functional interaction.

That's why it makes sense to cluster the results depending on the functional interaction. Every cluster is identified by a set of functional interactions. A larger cluster indicates a more task independent functional interaction which occurs in many different situations. In contrast, a smaller cluster indicates a more task dependent functional interaction which occurs in few contexts.

2.6 Implementation

The second part of the thesis reuses the implemented `WindowsAutomationAPI` which is written in C#. This comes from the fact that the UIA framework is easily accessible with C#. Foreign algorithms and frameworks are commonly written in Java. Thus, the thesis project subsequently has to call extern programs and has to communicate information with standardized formats. This is the case for strategy 1–3. However, the n-gram based strategy is implemented in C# without an extern library.

2.6.1 Strategy 1: Sequential Pattern Mining

Fournier-Viger et al. implement Sequential Pattern Mining Framework (SPMF) an open-source data mining library [31]. They offer 78 data mining algorithms for

- sequential pattern mining,
- association rule mining,
- frequent itemset mining,
- high-utility pattern mining,
- sequential rule mining and
- clustering.

However, only sequential pattern mining algorithms are considered for the strategy. In particular, the sequential pattern mining algorithms are categorized depending on the mined pattern type. The categories [32] distinguish between

- frequent sequential,
- closed sequential,
- maximal sequential,
- sequential generator,
- compressing sequential,
- top-k sequential and
- multidimensional sequential

patterns. The thesis focuses exclusively on maximal sequential patterns. The website provides a rich documentation on nearly every algorithm [33]. The VMSP algorithm is described in [33, example 59]. The author explains the usage of the algorithm as well as input and output file formats.

A special file format is used for communicating the sequence database [33, Input file format]. Every line represents a sequence. The items are positive numbers. Fortunately, generalized EOI IDs are always positive. The items in an itemset are separated by space. However, the sequence database of the thesis has only one-element sets. That's why the value "−1" succeeds every ID, which determines the end of an itemset. The value "−2" illustrates the end of a sequence. Listing 1 shows a sequence database based on generalized EOI IDs.

Listing 1: Sequence database in the SPMF input file format

```
49483 −1 64819 −1 −2
49444 −1 49483 −1 64819 −1 −2
49482 −1 82419 −1 −2
49482 −1 82419 −1 −2
66206 −1 49482 −1 −2
49485 −1 129693 −1 49330 −1 −2
49485 −1 129685 −1 −2
49484 −1 49330 −1 −2
66206 −1 66223 −1 −2
66206 −1 147190 −1 −2
147292 −1 49540 −1 −2
```

The sequence database is written to a file `input.txt`. The algorithm is invoked with the following example command: `java -jar ../spmf.jar run VMSP input.txt output.txt 10% 100`

The parameters have the following meanings: VMSP stores the resulting patterns in the `output.txt` file. The next command line parameter sets the minsup value to 10%. The last parameter assigns the maximal length to 100.

Listing 2 illustrates the statistics returned by the VMSP algorithm.

Listing 2: Example result statistics of the VMSP algorithm

```
============= Algorithm VMSP — STATISTICS =============
 Total time ~ 267 ms
 Frequent sequences count : 8
Max memory (mb) : 4.31258
minsup 1
 Intersection count 5416
==================================================
```

Besides the total computation time and the maximal memory usage, the statistics shows the quantity of frequent sequences and the absolute minimal support value. Moreover, the intersection count is shown. Intersections are calculated in the search procedure. Because intersections are costly, a small value is desirable.

A particular file format is used for communicating the results [33, Output file format]. It is very similar to the input file format with two differences: No "—2" value indicates the end of a sequence and after an additional "SUP:" value the support value is written. The format is parsed by the implementation for further postprocessing. Listing 3 depicts a result of the algorithm.

Listing 3: VMSP result in the SPMF output file format

```
82419 —1 SUP: 2
66206 —1 SUP: 3
49485 —1 SUP: 2
49330 —1 SUP: 2
49482 —1 82419 —1 SUP: 2
49483 —1 64819 —1 SUP: 2
```

2.6.2 Strategy 2: Graph Mining

The graph mining is implemented with the help of the Parallel and Sequential Graph Mining Suite (ParSeMiS) [42]. This framework, which is written in Java, "searches for frequent, interesting substructures in graph databases" [42]. Hence, a graph database has to be created.

To communicate graphs from C# to Java this thesis uses the GraphML [76] file format. It can be used to store graphs with flexible application-specific data. Besides the structure with nodes and edges, GraphML specifies additional information in scalar values with attributes [12, section 2.4]. An attribute has to be declared with an ID, a domain, a name and type. The types boolean, int, long, float, double, and string are supported. Attributes are defined inside the corresponding domain, which can be graph, node or edge.

In C# graphs are stored with the QuickGraph library [39]. It provides generic graph structures, algorithms, visualizations and serializations. The `GraphMLExtensions` class [38] implements the `SerializeToGraphML` method to serialize and `DeserializeFromGraphML` method to parse GraphML files. Listing 4 illustrates a serialized element graph.

Listing 4: Serialized graph in GraphML format

```
<?xml version="1.0" encoding="utf-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
    <key id="xml_id" for="node" attr.name="xml_id" attr.type="int" />
        <graph id="G" edgedefault="directed">
                <node id="49483">
                        <data key="xml_id">49483</data>
                </node>
                <node id="64819">
                        <data key="xml_id">64819</data>
                </node>
                <edge id="From49483To64819" source="49483" target="64819" />
        </graph>
</graphml>
```

A special attribute `xml_id` is declared for nodes to store the ID. This comes from the fact that ParSeMiS handles GraphML nodes and edges with `Map<String, String>`, a map between attribute keys and values. Thus, the entities can only be identified by their attributes.

The algorithm is applied as follows: First, the graph database is created by transforming transactions to element graphs. Only graphs with at least one edge are considered. Every graph is written to its own file in the GraphML file format. Second, ParSeMiS is customized to parse the files, form a graph database, mine and serialize the resulting fragments. Listing 5 shows how the ParSeMiS code is adopted.

Listing 5: Direct access to the ParSeMiS mining functionality

```
1  MapLabelParser mapLabelParser = new MapLabelParser();
2  GraphmlParser<Map<String, String>, Map<String, String>> graphmlParser = new
       GraphmlParser<>(mapLabelParser, mapLabelParser);
3  de.parsemis.graph.ListGraph.Factory<Map<String, String>, Map<String, String>> factory =
       new de.parsemis.graph.ListGraph.Factory<>(mapLabelParser, mapLabelParser);
4
5  //parsing omitted
6
7  Settings<Map<String, String>, Map<String, String>> settings = new Settings<>();
8  settings.algorithm = new de.parsemis.algorithms.gSpan.Algorithm<>();
9  settings.strategy = new BFSStrategy<>();
10 settings.graphs = graphs;
11 settings.factory = factory;
12 settings.minFreq = new IntFrequency(minFreq);
13 settings.minNodes = minNodes;
14 settings.minEdges = minEdges;
15 settings.connectedFragments = true;
16 settings.javaparty = false;
17 settings.threadCount = 1;
18
19 Collection<Fragment<Map<String, String>, Map<String, String>>> fragments = Miner.mine(
       graphs, settings);
20
21 //serialization omitted
```

Line 1-3 illustrates the usage of the parsing facility of ParSeMiS. The `MapLabelParser` (line 1) converts nodes and edges to `Map<String, String>` maps, which store defined GraphML attributes. A `GraphmlParser` (line 2) utilize the given label parser. The class can be used to parse and serialize the GraphML format. However, the parsing and serialization code is omitted because of pettiness. The factory (line 3) is used in the fragment (candidate) generation process.

Line 7-17 specifies the necessary settings for the mining step (line 19). Besides the algorithm, a search strategy for the fragments can be selected. The `graphs` variable contains the necessary graph database. Furthermore, the thresholds and small settings are assigned. The static `mine` method of the `Miner` class returns a collection of fragments, given the graph database and the settings. Every fragment is a subgraph and contains a frequency value.

Besides, the self-written code implements a simple argument parsing. Thus, the ParSeMiS code can in invoked in this way:

```
java -jar parsemis.jar <list of graphml files> gSpan 2 2 1 <output folder>
```

The algorithm settings follow after a list of GraphML file references and the algorithm name. The first setting argument describes the minimal frequency. This is followed by the minimal node and minimal edge threshold. Finally, an output folder specifies where the fragments are written.

In the end, the fragments are written to an output folder and parsed in C# for further postprocessing.

2.6.3 Strategy 3: Process Mining

Process Mining is implemented with ProM 6.4.1 [81]. This Open Source software, written in Java, contains various process mining algorithms. An introduction to ProM [82] helps to understand the basics of the tool. ProM consists of packages (plug-ins) which provides a lot of functionality [82, section 2.2]. They are downloaded separately at the first run of the tool. The application presents a GUI and has three main tabs [82, section 3]: (1) the workspace view, (2) the action view and (3) the view view. (1) The workspace view shows imported logs and results [82, section 3.1]. (2) The action view lists all available actions that can be made with some input data [82, section 3.2]. The actions are implemented process mining algorithms. The tool suggests automatically appropriate actions based on the input data type. (3) The view view shows different views of resources like logs and results [82, section 3.3].

The tutorial [84] demonstrates solutions of ProM to specific problems. The tutorial presents questions that can be answered by using ProM. An example event log helps to play with the tool. [84, Section 1.2] gives a concise introduction to process mining. [84, Section 3.2] focuses on mining case-related information about a process by using the Pattern Abstractions visualization.

The XES format is used to import self-generated event logs. An XML Schema Definition (XSD) of XES is downloadable, which is the "XML serialization of the XES format for event log data" [72]. With the help of Xsd2Code [14] the definition can be used to generate C# Business Entity classes. Thus, it's easy to instantiate classes and serialize them to Extensible Markup Language (XML).

After the event log is generated, an algorithm can be invoked on the data. The process mining strategy wants to extract pattern abstractions from an event log. This functionality is implemented in ProM as a package called *Pattern Abstractions*. ProM provides an easy-to-use GUI, but this thesis uses the underlying libraries and packages of ProM. The main tool and the basic libraries are distributed by the authors of ProM. However, packages are downloaded and stored separately in the default folder `.ProM64/packages` [82, section 2.3]. In this folder a file `packages.xml` describes the local package repository. A look into this file reveals an entry <repository url="http://www.promtools.org/prom6/packages641/PatternAbstractions/packages.xml"/>. The content of this file is visualized in Listing 6.

Listing 6: The packages.xml file of the Pattern Abstractions package

```
<packages>
        <package author="J.C.␣Bose" auto="false" desc="Pattern␣Abstractions" hasPlugins
            ="true" license="LGPL" logo="http://www.promtools.org/prom6/packages641/
            prom_subtitle_hat_300.png" name="PatternAbstractions" org="Eindhoven␣
            University␣of␣Technology" os="all" url="http://www.promtools.org/prom6/
            packages641/PatternAbstractions/PatternAbstractions-6.4.104-all.zip"
            version="6.4.104">
                <dependency name="BasicUtils"/>
                <dependency name="Log"/>
                <dependency name="InteractiveVisualization"/>
        </package>
</packages>
```

This investigation uncovers the download link of the package, which is `http://www.promtools.org/prom6/packages641/PatternAbstractions/PatternAbstractions-6.4.104-all.zip`. The ZIP file contains a Java Archive (JAR) file `PatternAbstractions.jar`, which again contains, among others, the two classes `MinePatterns.class` and `MineAbstractions.class`. Additionally, the source code is contained in the JAR file.

The `MinePatterns` class implements the discovery of maximal repeats. The class has to be instantiated with an already parsed log. The method `findPatterns` discovers patterns with respect to a given configuration. The pattern type (tandem arrays or maximal repeats) and the pattern length preference (short or long) can be configured. Mappings from sets to sets contain the results. The key is the repeat alphabet, while the value contains the patterns.

The `MineAbstractions` class implements the discovery of abstractions based on set theory. Given a pattern alphabet set the `mineSetTheoryAbstractions` method discovers the abstractions. The instance of the `AbstractionSetTheory` class contains the desired maximal elements.

Listing 7 presents the self-written Java code to use the functionality of the Pattern Abstractions plug-in on a programming level.

Listing 7: Direct access to the Pattern Abstractions plug-in functionality

```
1  UIContext ctx = new UIContext();
2  UIPluginContext pluginCtx = ctx.getMainPluginContext();
3  PatternAbstractionFrame frame = new PatternAbstractionFrame(pluginCtx, log);
4  MinePatterns minePatterns = new MinePatterns(frame, log);
5  Set<PatternType> patternTypes = new HashSet<>();
6  patternTypes.add(PatternType.MaximalRepeats);
7
8  minePatterns.findPatterns(patternTypes, PatternLengthPreference.Longer);
9
10 Map<TreeSet<String>, TreeSet<String>> papsm = minePatterns.
       getEntireLogMaximalRepeatPatternAlphabetPatternSetMap();
11 EncodedLog elog = minePatterns.getEncodedLog();
12
13 JPanel panel = new JPanel();
14 MineAbstractions mineAbstractions = new MineAbstractions(elog, panel);
15 mineAbstractions.setAbstractionStrategy(AbstractionStrategy.SetTheory);
16
17 Set<Set<String>> patternAlphabetSet = new HashSet<>();
18 patternAlphabetSet.addAll(papsm.keySet());
19
20 mineAbstractions.mineSetTheoryAbstractions(patternAlphabetSet);
21 AbstractionSetTheory ast = accessAST(mineAbstractions);
22 List<Set<String>> maximalElementList = ast.getMaximalElements();
23
24 List<Set<String>> abstractions = postprocess(maximalElementList, 2, minePatterns);
```

The variable `log` of type `XLog` contains a parsed event log. Line 1-3 are necessary to instantiate the `MinePatterns` class. Line 5-6 configures the pattern type for maximal repeats. In line 8 the pattern discovery algorithm is invoked with a preference for longer pattern lengths. The variable `papms` contains a mapping between the pattern alphabet and pattern set. The keys of this map are the repeat alphabets. An encoded log can be acquired in line 11. This log maps activities to a character-based representation. It's necessary for the instantiation of the `MineAbstractions` class in line 14. Line 15 sets the abstraction strategy to set theory. Line 17-18 allocates the set of pattern alphabets by using the keys of the `papms` map. The abstractions are mined in line 20 with the given pattern alphabet set. The attribute `ast` of the `MineAbstractions` class contains the maximal elements. However, this attribute is private and that's why it has to be accessed by reflection with the self-written `accessAST` method. Line 22 receives the maximal elements from the set theory. In a postprocessing step in line 24, the character-based representation is decoded to the initial activities. At the same time, only sets of size 2 are considered.

2.7 Evaluation

The evaluation is divided in the following steps: First, promising user-program pairs are determined which are considered for the subsequent steps. Second, the interaction time is analyzed. Third, parameters (setscrews) of the transaction identification approaches are studied. Fourth, the commonly used pattern mining setup is explained in detail. Fifth, reference patterns are compared with the mining results. Finally, mined patterns are analyzed by participants which evaluating the results with a Likert scale.

2.7.1 User-Program Pairs

The data contains 247 user-program pairs. However, only some pairs are suitable for mining patterns. The functional aspect is important for patterns. That's why the pairs are measured by their functional interactions. In particular, the count of distinct functional interactions determines the functional power. More distinct functional interactions can produce more distinct patterns. This analogy helps to find appropriate user-program pairs.

Table 7 shows a list of 25 promising user-program pairs which are sorted by the distinct functional interaction count. On the left side, the table shows participants working with a PC to interact with an application software. On the right side, the count of program interactions and user interactions are shown. While program interactions are only those interactions made with the application software, user interactions are all interactions observed from the user. The program interactions are shown with an absolute count as well as the relative proportion to all made interactions. In between, there are the seven classes, explained in section 2.5.2. All interactions, as well as, distinct interactions having

the class are counted. An interaction of the same class is counted twice, if it interacts with the same generalized EOI. In case of functional interactions (F), keyboard shortcuts are considered as well. A keyboard interaction of the functional class is counted twice, if the associated key is the same. Hence, the distinct functional interaction count shows maximal different used functionalities.

The first entry with the application software *javaw* is in fact the Eclipse IDE [30]. This application software provides a rich menu structure, buttons for various tasks and many shortcuts. Moreover, participant 8 is member of the power user group. That's why we found 82 distinct functional interactions.

The second and third entries demonstrates two different email programs from two different participants. They are located on top because these programs are high functionality applications. Besides deep menus for configuration, there are various email tasks including, among others, writing, sending, answering, moving and deleting emails. Additionally, an email program is at the same time a word processor when writing emails. That's why shortcuts are also possible functional interactions.

It attract attention that the application software *starmoney* is listed multiple times by the same participant. These entries reflect different versions of the same application software. During observation application softwares are updated which results in another program hash. That's why they are strongly distinguished by the system. On the other side, the application software *winword* is shown multiple times by various users. They all using different versions of the same application software. It seems that this application software contains a lot of functionalities, because it is listed under the top 10. Word processors are commonly used and contain functionalities which are usually accessed by various shortcuts. That's why *winword*, *notepad++* and *texstudio* are highly located.

The classification is based on the generalized EOI. No semi-structural-informative (SI) interactions are found. That's because it's unusual to interact with a separator element. Additionally, no informative (I) interactions are found. That's why all informative elements are generalized to their container elements (S). As expected, there are no semi-functional-structural (FS) interactions because no user interacted with a splitbutton. However, some semi-informative-functional (IF) interactions exists. This comes from the fact that application softwares let users often change the internal state by alterable elements. Some interactions in the special *None* class are found. These members reflect erroneous interactions as well as interactions with observer and process events. Power users perform the most functional interactions. Unfortunately, the distinct functional interaction counter drops rapidly. However, we continue the investigation with these pairs.

Part.	PC	App.	None		S		SI		I		IF		F		FS		Prog. Inter.		User Inter.
			all	dist.	all	dist.	all	dist.	all	dist.	all	dist.	all	dist.	all	dist.	abs.	rel.	
Participant 8	8224	javaw	160	0	163	41	0	0	0	0	110	14	238	82	0	0	671	17.69%	3794
Participant 8	8224	outlook	379	4	194	13	0	0	0	0	106	25	409	77	0	0	1088	28.68%	3794
Participant 6	0B8C	thunderbird	197	0	937	33	0	0	0	0	113	54	448	45	0	0	1695	42.88%	3953
Participant 1	F52C	winword	152	0	8	3	0	0	0	0	0	0	86	33	0	0	246	5.43%	4531
Participant 8	8224	winword	139	0	21	8	0	0	0	0	45	11	181	33	0	0	386	10.17%	3794
Participant 1	F52C	notepad++	254	0	26	1	0	0	0	0	5	1	84	27	0	0	369	8.14%	4531
Participant 1	F52C	pgadmin3	152	0	42	5	0	0	0	0	3	3	50	26	0	0	247	5.45%	4531
Participant 7	3CF1	winword	331	0	19	5	0	0	0	0	2	1	58	25	0	0	410	25.98%	1578
Participant 1	F52C	texstudio	73	0	1047	2	0	0	0	0	2	1	617	22	0	0	1739	38.38%	4531
Participant 4	85EC	dllhost	52	0	3	1	0	0	0	0	0	0	165	21	0	0	220	33.23%	662
Participant 1	F52C	acrord32	121	0	4	2	0	0	0	0	11	2	45	21	0	0	181	3.99%	4531
Participant 6	0B8C	starmoney	125	0	9	6	0	0	0	0	6	6	26	20	0	0	166	4.20%	3953
Participant 6	0B8C	starmoney	169	0	8	5	0	0	0	0	57	21	54	20	0	0	288	7.29%	3953
Participant 6	0B8C	starmoney	124	0	7	5	0	0	0	0	18	13	33	19	0	0	182	4.60%	3953
Participant 6	0B8C	winwein	94	0	16	2	0	0	0	0	10	1	26	18	0	0	146	3.69%	3953
Participant 8	8224	powerpnt	422	0	1	1	0	0	0	0	0	0	50	18	0	0	473	12.47%	3794
Participant 6	0B8C	winwein	189	0	8	3	0	0	0	0	4	1	34	17	0	0	235	5.94%	3953
Participant 8	8224	keepass	74	0	2	2	0	0	0	0	1	1	22	16	0	0	99	2.61%	3794
Participant 8	8224	pdfxcview	153	0	5	3	0	0	0	0	0	0	19	15	0	0	177	4.67%	3794
Participant 4	85EC	mspaint	29	0	3	3	0	0	0	0	0	0	20	15	0	0	52	7.85%	662
Participant 1	F52C	outlook	115	0	27	10	0	0	0	0	3	2	47	15	0	0	192	4.24%	4531
Participant 8	8224	calc	65	0	0	0	0	0	0	0	0	0	35	14	0	0	100	2.64%	3794
Participant 7	3CF1	acrobat	180	0	9	4	0	0	0	0	158	62	82	14	0	0	429	27.19%	1578
Participant 7	3CF1	dllhost	100	0	0	0	0	0	0	0	0	0	89	14	0	0	189	11.98%	1578
Participant 6	0B8C	starmoney	106	0	5	2	0	0	0	0	23	10	37	14	0	0	171	4.33%	3953

Table 7: The 25 promising user-program pairs and their classifications sorted by distinct functional interactions

2.7.2 Interaction Time

Because two transaction identification approaches are based on time constrains, we first investigate the interaction time. Table 8 shows the participants, using application softwares, with their interaction time. The time spans are the differences of the time stamps of all consecutive interactions. The columns show the minimal and maximal time spans as well as mean and standard deviation of all time spans. On the right side, the program interaction count is shown again.

The minimal interaction time gives information about the usage of the application software. Power user often tends to small interaction times. However, slowly responding application softwares can raise the minimal interaction time. The data reveals that time windows of 30 seconds or higher should be used to receive long enough transactions.

In contrast, the maximal interaction time shows long breaks which continue about a week. However, these are individual cases. They occur because of traveling and holidays. The breaks can be easily determined with the supporting transaction identification approaches. The mean demonstrates that these extreme rests are individual cases.

The mean of the interaction times expresses the average time span between the interactions. A short mean value indicates a user who works day and night with the application software. In contrast, a long mean value argues longer breaks between the application software usage.

The standard deviation indicates the deviations from the mean. The values express that we found very long breaks as well as very short breaks.

The Gantt diagrams reveal that users working in a certain period with short interaction times. After the work is done very long interaction times occur because of breaks. For example, the Gantt chart in figure 4 shows seven days of participant 8 working with various application softwares per line.

Part.	App.	Min.	Mean	Std. Dev.	Max.	Prog. Inter.
Participant 8	javaw	00:00:05	01:55:18.4670000	12:01:07.4490000	7.22:06:26	671
Participant 8	outlook	00:00:01	01:11:06.5250000	09:10:50.0790000	7.02:53:21	1088
Participant 6	thunderbird	00:00:11	00:59:24.0380000	05:54:43.1070000	6.14:50:17	1695
Participant 1	winword	00:00:02	03:08:15.0860000	14:45:49.7280000	6.20:44:38	246
Participant 8	winword	00:00:03	03:20:39.9300000	15:36:33.9480000	7.06:22:36	386
Participant 1	notepad++	00:00:01	02:05:29.2010000	12:07:12.3660000	6.20:44:38	369
Participant 1	pgadmin3	00:00:02	03:07:29.1710000	14:45:16.8860000	6.20:44:38	247
Participant 7	winword	00:00:16	02:26:36.8880000	11:35:37.4240000	7.22:56:38	410
Participant 1	texstudio	00:00:02	00:26:32.2300000	05:37:38.1030000	6.20:44:38	1739
Participant 4	dllhost	00:00:04	04:17:23.5110000	22:23:20.4660000	9.05:28:29	220
Participant 1	acrord32	00:00:02	04:16:13.8670000	17:05:17.8650000	6.20:44:38	181
Participant 6	starmoney	00:00:21	10:09:19.4670000	18:29:35.6100000	6.22:56:38	166
Participant 6	starmoney	00:00:17	05:50:18.5090000	14:48:16.6930000	6.22:56:38	288
Participant 6	starmoney	00:00:11	09:15:27.6910000	17:57:50.1490000	6.22:56:38	182
Participant 6	winwein	00:00:06	11:33:22.1520000	19:26:10.2860000	6.22:56:38	146
Participant 8	powerpnt	00:00:04	02:43:40.7060000	14:43:41.5760000	7.22:06:26	473
Participant 6	winwein	00:00:07	07:09:39.1110000	16:14:03.3430000	6.22:56:38	235
Participant 8	keepass	00:00:05	13:08:19.7240000	1.04:50:17.5610000	7.02:51:24	99
Participant 8	pdfxcview	00:00:03	07:18:57.3470000	22:30:47.8940000	7.05:48:50	177
Participant 4	mspaint	00:00:09	18:24:21.0200000	1.19:37:11.4690000	9.05:28:29	52
Participant 1	outlook	00:00:02	04:02:13.3870000	16:36:02.6280000	6.20:44:38	192
Participant 8	calc	00:00:05	13:00:21.9490000	1.05:57:52.5240000	7.22:06:26	100
Participant 7	acrobat	00:00:04	02:21:29.1680000	11:22:57.0020000	7.22:56:38	429
Participant 7	dllhost	00:00:05	05:18:57.9100000	16:45:45.3440000	7.22:53:50	189
Participant 6	starmoney	00:00:15	09:51:24.1880000	18:13:13.1960000	6.22:56:38	171

Table 8: The 25 user-program pairs with regard to their interaction time

2.7.3 Transaction Identification

In the third step, we evaluate the transaction identification approaches (TI). This step will analyze appropriate parameter values.

The tables below will always show the 25 user-program pairs, however with different transaction identification results. The result is presented with the following measurements: The *Min.* column shows the minimal length, while the *Max.* column reveal the maximal length of the transactions. In between, the mean gives an average measure of the length, while the standard deviation (*Std. Dev.*) indicates how divergent the length is from the mean value. The *Count* column expresses how many transactions are identified.

In some cases the values have to be written in one cell. Then, the five measurements are separated by an slash symbol "/" in the following order:

Min. / Mean / Std. Dev. / Max. / Count

(TI1) In the case of the first transaction identification approach no parameter is necessary. A transaction is identified if it ends with a functional interaction. However, we will investigate the results in table 9.

The minimal length of a transaction stays one. There are always a sequence where two consecutive interactions are classified functional. In this case they are separated (see section 2.5.2 (TI1)). In particular, a sequence of a button click and an immediately shortcut occurs frequently. However, two consecutive button clicks are also not unlikely.

In contrast, there are various maximal length values of the transactions. However, the maximal values never exceed a value of 13. There are no outliers with absurd lengths. This comes from the fact that these selected pairs contain many different functional interactions. Thus, the probability, that no functional interaction occurs, is low.

The mean length stays stable between approximately 1.5. Moreover, the standard deviation is small. This results in homogeneous transactions. However, they are in average very short.

The quantity of the transactions rarely exceeds a value of 50. This prophecies insufficient data for finding patterns based on frequency. However, longer transactions would result in fewer transactions.

(TI2) The second transaction identification approach is based on the relative frequency of navigation interactions. The frequency can range between the two extremes 0.0 and 1.0. Table 10 visualizes the transaction identification outcome on different parameter values.

The lowest value 0.0 assumes that no navigation interactions exist. Every interaction is believed to be a content interaction. That's why the minimal and maximal length stays one. Each transaction consists of exactly one interaction. In contrast, the highest value 1.0 assumes that every interaction is a navigation transaction. Hence, no content interaction exists which would identify a transaction. That's why the quantity stays one and minimal and maximal length are equal.

In between, an appropriate value has to be determined. The value is appropriate if the following criteria are satisfied: (1) The minimal length should be more then one to encourage longer patterns. (2) The maximal length should be approximately the expected pattern size. (3) The mean should be in the middle between minimal and maximal length. Thus, the transactions are homogeneous. (4) The standard deviation should be low to disallow outliers. (5) The quantity of the transactions should be at the same time high to have enough transactions for mining patterns.

Actually, the best value satisfying these criteria is approximately 0.4. While for higher values the transaction quantity decreases, the lower value 0.2 lacks in long enough transactions in average.

(TI3) The third transaction identification approach has no decimal parameter. However, two clues are used to determine the maximal forward reference. In this evaluation we will investigate what clue works best for identifying appropriate transactions.

Table 11 contrasts the two clues. For each pair the two methods are measured with the minimal and maximal length as well as mean and standard deviation, and finally the quantity of the transactions. Additionally, the average of the values are located at the bottom of the table.

In comparison, both clues have approximately the same minimal length, standard deviation and maximal length in average. However, the average mean and average count differs noticeable. The maximal forward crawl method has an average mean length of 2.1. In contrast, the maximal forward element method has an average mean length of 3.1. While the former method identifies in average 40 transactions, the latter method returns in average only 35.2. This comes from the fact that backward crawls occur more likely then backward elements. The result is a shorter transaction length and more transactions are identified.

In conclusion, the maximal forward crawl makes intuitively more sense. Indeed, crawls can be interpreted as pages containing elements. It is true that this approach returns shorter transactions, however they still have an acceptable size. Moreover, the count of identified transactions could positively influence the mining result.

(TI4) The last transaction identification approach strongly depends on time. Depending on the size of the time window different transactions are identified. Table 12 shows the results for some values. The time window is expressed in seconds and range from 30 seconds to 180 seconds in a 30 second interval.

The smallest time window of 30 seconds identifies very short transactions. They are in average close to 1. The maximal transaction size never exceeds the value of 4. This signals that the user do very few interactions in this period of time. The largest time window of 180 seconds identifies not much larger transactions. The standard deviation is still quite small. However, the count of the transactions drops. For slow interacting users the larger time window doesn't change the results drastically.

Although the time window ranges from half a minute to three minutes, the measurements change slightly. Because the standard deviance stays small, the transactions are very homogeneous. Hence, this approach is stable for different time window values.

Part.	App.	Min	Mean	Std. Dev.	Max	Count
Participant 8	javaw	1	2.3	1.9	13	141
Participant 8	outlook	1	2.0	1.5	8	217
Participant 6	thunderbird	1	2.8	1.7	13	423
Participant 1	winword	1	1.2	1.0	6	25
Participant 8	winword	1	1.7	1.3	8	44
Participant 1	notepad++	1	1.6	1.1	7	40
Participant 1	pgadmin3	1	1.9	1.4	6	40
Participant 7	winword	1	1.6	1.0	6	28
Participant 1	texstudio	1	2.4	2.5	7	9
Participant 4	dllhost	1	1.0	0.2	2	63
Participant 1	acrord32	1	1.1	0.3	2	22
Participant 6	starmoney	1	1.6	0.8	4	20
Participant 6	starmoney	1	2.3	1.3	7	45
Participant 6	starmoney	1	1.9	1.0	4	26
Participant 6	winwein	1	1.7	2.2	11	23
Participant 8	powerpnt	1	1.0	0.0	1	6
Participant 6	winwein	1	1.2	0.7	5	32
Participant 8	keepass	1	1.2	0.4	2	19
Participant 8	pdfxcview	1	1.4	0.8	4	14
Participant 4	mspaint	1	1.2	0.4	2	20
Participant 1	outlook	1	2.0	1.5	8	25
Participant 8	calc	1	1.0	0.0	1	33
Participant 7	acrobat	1	2.5	1.3	9	67
Participant 7	dllhost	1	1.0	0.0	1	36
Participant 6	starmoney	1	2.0	1.0	5	25

Table 9: The 25 user-program pairs with regard to the first transaction identification approach (TI1)

Part.	App.	Relative Frequency of Navigation Interactions					
		0.0	0.2	0.4	0.6	0.8	1.0
Participant 8	javaw	1/1.0/0.0/1/141	1/1.1/0.3/2/134	1/2.4/2.4/14/90	1/4.4/5.2/31/57	1/8.7/10.3/59/34	331/331.0/0.0/331/1
Participant 8	outlook	1/1.0/0.0/1/217	1/1.3/1.1/10/193	1/2.2/2.1/13/143	1/4.0/3.7/23/95	1/5.4/4.8/23/76	448/448.0/0.0/448/1
Participant 6	thunderbird	1/1.0/0.0/1/423	1/1.2/0.8/11/411	1/1.8/1.8/19/363	1/3.3/3.1/27/282	1/5.7/5.9/49/187	1178/1178.0/0.0/1178/1
Participant 1	winword	1/1.0/0.0/1/25	1/1.4/1.1/6/20	1/2.0/2.2/10/16	1/2.8/3.5/14/12	1/3.7/5.5/19/9	33/33.0/0.0/33/1
Participant 8	winword	1/1.0/0.0/1/44	1/1.8/1.5/8/31	1/3.7/3.8/17/19	1/3.8/3.8/17/19	1/4.3/4.2/17/17	79/79.0/0.0/79/1
Participant 1	notepad++	1/1.0/0.0/1/40	1/1.7/1.4/8/32	1/2.6/3.0/15/24	1/3.4/5.4/26/19	1/5.0/9.3/37/13	67/67.0/0.0/67/1
Participant 1	pgadmin3	1/1.0/0.0/1/40	1/1.0/0.0/1/40	1/1.4/0.8/4/35	1/2.5/2.0/8/24	1/5.1/3.0/12/14	77/77.0/0.0/77/1
Participant 7	winword	1/1.0/0.0/1/28	1/1.3/0.6/3/25	1/1.7/1.2/5/22	1/1.9/1.2/5/22	1/1.9/1.2/5/22	46/46.0/0.0/46/1
Participant 1	texstudio	1/1.0/0.0/1/9	1/1.6/0.9/3/8	1/2.0/1.1/4/7	1/2.8/1.5/5/5	2/3.2/1.2/5/5	42/42.0/0.0/42/1
Participant 4	dllhost	1/1.0/0.0/1/63	1/2.1/3.1/18/31	1/2.4/3.3/18/27	1/2.7/3.5/18/24	1/3.6/4.3/18/18	65/65.0/0.0/65/1
Participant 1	acrord32	1/1.0/0.0/1/22	1/1.2/0.7/4/18	1/1.6/1.1/5/15	1/2.2/1.6/6/13	1/2.9/1.6/6/10	29/29.0/0.0/29/1
Participant 6	starmoney	1/1.0/0.0/1/20	1/1.1/0.4/3/20	1/1.3/1.2/6/18	1/2.5/1.6/6/13	1/4.0/2.5/9/8	35/35.0/0.0/35/1
Participant 6	starmoney	1/1.0/0.0/1/45	1/1.1/0.2/2/45	1/1.8/1.7/9/39	1/2.5/2.1/9/34	1/6.5/9.3/40/16	105/105.0/0.0/105/1
Participant 6	starmoney	1/1.0/0.0/1/26	1/1.8/1.5/6/20	1/2.4/3.4/14/15	1/2.6/5.1/21/14	1/4.5/5.7/21/10	50/50.0/0.0/50/1
Participant 6	winwein	1/1.0/0.0/1/23	1/1.5/0.9/4/16	1/2.6/2.0/7/10	1/5.0/3.5/12/8	1/6.7/3.6/12/6	40/40.0/0.0/40/1
Participant 8	powerpnt	1/1.0/0.0/1/6	1/1.0/0.0/1/6	1/1.5/0.9/3/4	1/2.0/1.4/4/3	1/2.0/1.4/4/3	7/7.0/0.0/7/1
Participant 6	winwein	1/1.0/0.0/1/32	1/1.6/1.0/5/22	1/2.3/2.0/8/17	1/3.3/3.2/11/12	1/3.9/3.3/11/10	39/39.0/0.0/39/1
Participant 8	keepass	1/1.0/0.0/1/19	1/1.7/0.8/3/13	1/2.0/1.0/4/11	1/2.0/1.0/4/11	1/2.2/1.2/4/10	22/22.0/0.0/22/1
Participant 8	pdfxcview	1/1.0/0.0/1/14	1/2.1/2.0/7/8	1/2.1/2.0/7/8	1/2.1/2.0/7/8	1/2.3/1.9/7/8	19/19.0/0.0/19/1
Participant 4	mspaint	1/1.0/0.0/1/20	1/1.1/0.2/2/19	1/2.4/2.8/10/9	1/3.7/5.1/15/6	2/7.7/7.3/18/3	23/23.0/0.0/23/1
Participant 1	outlook	1/1.0/0.0/1/25	1/1.3/0.7/4/24	1/1.8/1.7/8/21	1/2.5/2.2/10/18	1/3.7/2.6/10/14	52/52.0/0.0/52/1
Participant 8	calc	1/1.0/0.0/1/33	1/1.0/0.0/1/33	1/1.0/0.0/1/33	1/2.8/3.4/12/12	3/6.6/3.1/12/5	33/33.0/0.0/33/1
Participant 7	acrobat	1/1.0/0.0/1/67	1/1.3/0.8/5/64	1/3.2/4.9/33/42	1/6.8/18.7/90/21	1/9.9/23.8/101/16	184/184.0/0.0/184/1
Participant 7	dllhost	1/1.0/0.0/1/36	1/1.7/1.7/9/21	1/2.3/2.2/9/16	1/3.0/2.9/10/12	1/3.6/3.4/10/10	36/36.0/0.0/36/1
Participant 6	starmoney	1/1.0/0.0/1/25	1/1.2/0.5/3/21	1/1.2/0.5/3/21	1/2.9/1.9/7/15	1/5.2/5.4/17/9	50/50.0/0.0/50/1

Table 10: The 25 user-program pairs with regard to the second transaction identification approach (TI2)

Part.	App.	Maximal Forward Crawl					Maximal Forward Element				
		Min.	Mean	Std. Dev.	Max.	Count	Min.	Mean	Std. Dev.	Max.	Count
Participant 8	javaw	1	6.3	5.7	28	44	1	4.0	1.9	10	68
Participant 8	outlook	1	3.2	3.0	18	116	1	3.6	2.4	14	98
Participant 6	thunderbird	1	2.4	1.4	10	327	1	2.6	1.1	7	350
Participant 1	winword	1	2.8	3.5	14	12	1	2.7	1.9	7	11
Participant 8	winword	1	1.7	0.9	4	33	1	2.8	1.5	7	25
Participant 1	notepad++	1	1.5	0.7	4	33	1	2.2	1.2	5	25
Participant 1	pgadmin3	1	2.3	1.6	8	27	2	4.1	1.5	7	14
Participant 7	winword	1	1.3	0.7	4	27	1	2.2	1.2	5	19
Participant 1	texstudio	1	1.0	0.0	1	9	1	1.0	0.0	1	9
Participant 4	dllhost	1	1.1	0.3	2	57	1	1.7	0.9	4	39
Participant 1	acrord32	1	1.0	0.0	1	22	1	1.7	0.9	4	15
Participant 6	starmoney	1	2.9	1.4	5	11	1	6.4	4.5	12	5
Participant 6	starmoney	1	2.3	1.7	8	35	1	4.3	2.7	15	24
Participant 6	starmoney	1	1.7	1.4	7	20	1	5.6	3.4	12	9
Participant 6	winwein	1	2.4	1.9	9	14	1	3.5	1.9	7	8
Participant 8	powerpnt	1	2.0	1.4	4	3	1	1.8	0.4	2	4
Participant 6	winwein	1	1.4	0.8	4	24	1	1.8	1.0	4	20
Participant 8	keepass	1	1.6	0.6	3	13	1	2.0	1.1	4	11
Participant 8	pdfxcview	1	1.4	0.8	3	12	1	1.9	1.0	4	9
Participant 4	mspaint	1	2.1	0.8	4	11	4	5.8	1.3	7	4
Participant 1	outlook	1	2.4	2.3	10	18	1	2.9	1.5	6	14
Participant 8	calc	1	1.0	0.2	2	32	1	4.1	1.9	7	8
Participant 7	acrobat	1	2.6	1.8	10	48	1	2.6	1.3	8	55
Participant 7	dllhost	1	1.1	0.3	2	32	1	1.4	0.6	3	25
Participant 6	starmoney	1	1.9	1.0	5	20	1	3.9	1.6	6	12
		1.0	2.1	1.4	6.8	40.0	1.2	3.1	1.5	6.7	35.2

Table 11: The 25 user-program pairs with regard to the third transaction identification approach (TI3)

Part.	App.	Time Window in Seconds					
		30	60	90	120	150	180
Participant 8	javaw	1/1.1/0.3/2/133	1/1.5/0.5/3/125	1/1.8/0.7/3/113	1/2.1/0.8/4/105	1/2.4/0.9/5/100	1/2.5/1.2/6/94
Participant 8	outlook	1/1.1/0.2/2/212	1/1.2/0.4/4/199	1/1.3/0.6/5/189	1/1.5/0.7/5/180	1/1.6/0.8/6/172	1/1.8/0.9/7/165
Participant 6	thunderbird	1/1.0/0.1/2/423	1/1.0/0.2/3/421	1/1.1/0.4/3/415	1/1.2/0.4/3/412	1/1.3/0.5/4/403	1/1.3/0.6/4/401
Participant 1	winword	1/1.2/0.4/2/22	1/1.2/0.5/3/22	1/1.3/0.6/3/21	1/1.4/0.7/4/19	1/1.5/0.9/5/19	1/1.6/1.2/6/17
Participant 8	winword	1/1.2/0.5/3/39	1/1.4/0.6/3/35	1/1.6/0.7/3/33	1/1.9/0.9/4/29	1/2.0/0.8/3/28	1/2.3/1.1/5/27
Participant 1	notepad++	1/1.3/0.4/2/37	1/1.5/0.6/3/31	1/1.8/1.0/5/32	1/1.9/1.0/5/29	1/2.1/1.4/7/30	1/2.3/1.5/7/27
Participant 1	pgadmin3	1/1.2/0.4/2/37	1/1.3/0.5/2/36	1/1.7/0.7/3/33	1/1.9/0.9/4/28	1/1.9/1.0/4/27	1/2.3/1.3/6/27
Participant 7	winword	1/1.0/0.0/1/28	1/1.2/0.4/2/26	1/1.2/0.4/2/26	1/1.3/0.5/3/25	1/1.3/0.5/3/26	1/1.3/0.5/2/27
Participant 1	texstudio	1/1.1/0.3/2/9	1/1.3/0.5/2/9	1/1.5/0.5/2/8	1/1.6/0.7/3/8	1/1.6/0.9/3/8	1/1.8/0.8/3/8
Participant 4	dllhost	1/1.5/0.8/4/43	1/1.7/1.1/4/39	1/1.8/1.4/6/36	1/1.9/1.8/8/35	1/2.0/2.0/10/33	1/2.1/2.1/11/31
Participant 1	acrord32	1/1.0/0.0/1/22	1/1.0/0.0/1/22	1/1.2/0.4/2/19	1/1.2/0.4/2/19	1/1.2/0.4/2/20	1/1.2/0.4/2/19
Participant 6	starmoney	1/1.1/0.2/2/20	1/1.1/0.4/3/20	1/1.2/0.5/3/20	1/1.2/0.4/2/20	1/1.4/0.8/4/18	1/1.5/0.8/4/17
Participant 6	starmoney	1/1.1/0.2/2/45	1/1.4/0.5/2/44	1/1.6/0.7/3/42	1/1.8/0.8/4/41	1/1.8/0.9/4/39	1/2.1/1.2/5/36
Participant 6	starmoney	1/1.2/0.4/2/26	1/1.5/0.6/3/24	1/1.6/0.8/3/22	1/1.6/1.0/4/22	1/1.7/1.4/6/22	1/1.9/1.5/6/20
Participant 6	winwein	1/1.6/0.9/4/14	1/2.3/1.1/4/11	1/2.3/1.0/4/12	1/2.7/1.4/5/10	1/3.3/1.9/7/8	1/3.5/1.7/7/8
Participant 8	powerpnt	1/1.0/0.0/1/6	1/1.2/0.4/2/5	1/1.2/0.4/2/5	1/1.2/0.4/2/5	1/1.5/0.9/3/4	1/1.5/0.5/2/4
Participant 6	winwein	1/1.4/0.5/2/25	1/1.4/0.8/4/24	1/1.8/1.3/5/21	1/1.9/1.4/6/18	1/2.0/1.5/6/18	1/2.2/1.7/7/18
Participant 8	keepass	1/1.6/0.7/3/14	1/1.7/0.7/3/13	1/1.8/0.8/3/12	1/1.8/0.8/3/12	1/2.0/1.0/4/11	1/2.0/1.0/4/11
Participant 8	pdfxcview	1/1.2/0.4/2/12	1/1.5/0.7/3/10	1/1.4/0.7/3/10	1/1.7/0.8/3/10	1/1.9/1.1/4/9	1/1.9/1.1/4/9
Participant 4	mspaint	1/1.4/0.5/2/16	1/1.4/0.5/2/16	1/2.0/1.0/4/11	1/2.4/1.0/4/9	1/2.8/1.4/5/8	2/3.1/1.6/6/7
Participant 1	outlook	1/1.3/0.5/3/24	1/1.3/0.5/3/24	1/1.5/0.8/4/24	1/1.5/0.8/4/24	1/1.6/0.9/5/22	1/1.7/1.0/5/21
Participant 8	calc	1/2.8/1.0/4/12	3/4.7/1.4/7/7	1/5.5/3.3/11/6	5/8.3/2.5/12/4	5/8.3/2.5/12/4	5/8.3/4.1/15/4
Participant 7	acrobat	1/1.2/0.4/3/66	1/1.7/0.6/4/64	1/2.0/0.9/5/57	1/2.4/1.1/5/53	1/2.8/1.3/6/49	1/2.9/1.5/7/44
Participant 7	dllhost	1/1.3/0.5/3/28	1/1.6/0.9/4/22	1/1.7/1.2/5/21	1/2.0/1.4/6/18	1/2.3/1.7/7/16	1/2.3/1.9/8/16
Participant 6	starmoney	1/1.1/0.3/2/22	1/1.2/0.5/3/21	1/1.6/0.6/3/21	1/1.7/0.6/3/21	1/1.9/0.7/3/19	1/2.2/0.8/4/18

Table 12: The 25 user-program pairs with regard to the fourth transaction identification approach (TI4)

2.7.4 Pattern Mining Setup with Reference Pattern Analysis

The reference patterns reveal at the beginning of the research how patterns appear in real data. This section shows that they can be found by the discussed strategies. However insufficient interactions make it difficult to discover these patterns.

Participant 1 annotated six reference patterns in context of SourceTree 1.6.4.0, pgAdmin 1.18.1 and Outlook 14.0.7113.5000.

Git Pull is a pattern that consists of an initial invocation and a confirmation. Optional settings are configured in between. S1 with TI1 couldn't find that pattern. That's why the first button interaction is separated from the confirmation. However, S1 with TI2 and TI4 discovers this reference pattern. S1 with TI3 discovers a longer sequence with irrelevant interactions. This indicates that the transactions of TI3 are too long. S2 is more stable and discovers two node graphs where the reference pattern is among them. S3 performs similarly and returns appropriate sets with two elements. S4 reveals 2 instances of the 0-skip-2-gram. However, no strategy could discover the complexer pattern with additional settings.

Git Fetch is similar to Git Pull. Every time Git Pull is discovered, Git Fetch is mined, too.

Git Commit is a variable pattern with arbitrary many interactions with a tab item. However, it is finished with a shortcut. S1 and S2 are not appropriate for discovering patterns with shortcuts, because they focus on the generalized EOI. However, if we consider instead the generalized EOI, a pattern is found with 23 *Tab-66206* elements ending with a *Edit-66223* element. Because of infrequency S2 returns no appropriate graph. S3 couldn't find a suitable abstraction, too. S4 fails also because the reference pattern is infrequent.

Discard File Changes is a pattern that first configures settings in arbitrary interactions with tab items, then invokes the desired functionality and finally confirms it. S1 with T1 returns very long sequences which contains 15 *Tab-66206* and ends with *Button-66262 Block verwerfen*. Because TI1 splits at functional interactions with buttons, very long transactions with non-functional interactions are possible. However, the last confirmation is severed. S1 with T2 and T4 return only small patterns that indicate that reference pattern: *Button-69139, Tab-66206, Button-67057 Block verwerfen, Button-69139*. However, the confirmation button is believed to be the beginning. Only S1 with T3 returns the appropriate pattern: *Tab-66206, Tab-66206, Tab-66206, Tab-66206, Tab-66206, Tab-66206, Button-66262 Block verwerfen, Button-69139*. The absolute support is only 1. That's why the graph mining approach couldn't discover this reference pattern. The same holds for S3. In case of S4, only the skips made some suitable skip-grams frequent. For example the 1-skip-3-gram; *Mouse Left Click Tab-66206, Mouse Left Click Button-69139, Mouse Left Click Tab-66206*.

Show Data Table is a pattern that consists of three interactions. The first interaction specifies the target entity. A menu item leads to a final menu item, which invokes the desired functionality. While S1 with T1 and T2 couldn't discover that reference pattern, T3 returns a beautiful result: *Tree-51199, MenuItem-51351 Daten anzeigen, MenuItem-51359 Die obersten (100) Zeilen zeigen*. S1 with T4 returns the smaller exemplars: *MenuItem-51351 Daten anzeigen, MenuItem-51359 Die obersten (100) Zeilen zeigen* and *MenuItem-51351 Daten anzeigen, MenuItem-51360 Die letzten (100) Zeilen zeigen*. This indicates that the transactions have to be correct to discover patterns. S2 returns similar results. S3 with TI3 contains one abstraction that has the three interactions. S4 discovers 3 0-skip-3-grams that is the reference pattern.

Send Email illustrates that there are many possibilities why an email is sent. The possibilities always and with an invocation interaction, but starts different. In between there are various interactions. S1 with TI2 and TI4 discovers the small sequence *Button-39269 Antworten, Button-104402 Senden*. Because of infrequency S2 couldn't discover a similar pattern. This holds for S3, too. No skip-grams reflect the reference pattern. It's because the send possibilities are high, which results in variable infrequent sequences.

Participant 6 annotated four reference patterns occurring in Thunderbird 24.6.0.

Delete Email is a pattern with two interactions. The first interaction specifies what email will be deleted. The second interaction defines the delete operation. Thanks to the generalization data items are generalized to the corresponding table. S1 returns with TI1 the pattern *Table-2637, MenuItem-4249 Löschen* with a support of 100. The other transaction identification approaches (TIs) contain the pattern too, however with a smaller support and sometimes additional interactions. S2 behaves nearly the same. S3 contains abstractions { *Mouse Right Click Table-2637, Mouse Left Click MenuItem-163868 Löschen* } in different TIs. S4 discovers 99 0-skip-2-grams which represents the correct reference pattern.

Empty Trash is a pattern with three interactions. The first interaction selects the trash. The second interaction invokes the empty operation, while the third confirms that action. S1 couldn't find this pattern. This comes from the fact that the reference pattern appears too infrequent. This holds also for S2. S3 finds abstractions containing interactions of the reference pattern. It couldn't find a set with three interactions, however a greater set { *Mouse Left Click MenuItem-90856 Papierkorb leeren, Mouse Left Click Button-90866 Ja, Mouse Right Click Tree-2627, Mouse Left Click Tree-2627* } . S4 discovers 2 instances of the correct 3-gram, for example, *9719 (S) Mouse Right Click Tree-2627, 9720 (F) Mouse Left Click MenuItem-90856 Papierkorb leeren, 9721 (F) Mouse Left Click Button-90866 Ja*.

READ EMAIL is a pattern with arbitrary interactions. However, the first interaction opens the email, while the last interaction closes it. S1 discovers patterns with only the first and last interaction. Sometimes the order is reversed due wrong transaction identification. S2 presents many permutations of *Table-2637* and *Button-2998*. S3 discovers always a bigger set that contains both interactions. S4 returns 97 0-skip-2-grams of *Mouse Left Click Button-2988, Mouse Right Click Table-2637*. It's very frequent that after closing an email the table is used again. If we raise the skips we still can't find a pattern starting with table and ending with the close button. The variety of possibilities makes it difficult to detect that pattern.

RETRIEVE EMAIL is a sequence of functional interactions to retrieve emails. The pattern is so infrequent that no strategy could discover it. This comes from the fact that the program automatically searches for new emails.

Participant 8 annotated some coarse-granular patterns, however only one fine-grain reference pattern of Word 2013 was suitable.

PRINT DOCUMENT is a pattern that contains a shortcut to start the task. The next two interactions selects the printer. The last interaction starts the printing. Because there are insufficient interactions only 8 patterns are discovered. Only S4 discovers two patterns about printing. The first pattern *32740 (F) Mouse Left Click Button-145881 Drucken, 32747 (F) Keyboard Control + P* is a 2-gram that occurred two times. However, the ordering is unusual. The second pattern *32717 (IF) Mouse Left Click Edit-81486 1-Seiteninhalt, 32734 (F) Keyboard Control + P* shows the beginning of the reference pattern. The insufficient interactions and the infrequent reference pattern makes it difficult for discovery.

The reference patterns indicate what parameters work best to discover them. The mining setup is configured to find at least the reference patterns. This setup is commonly used for all 25 user-program pairs. The outcome is analyzed in the next section.

The settings of the strategies are as follows: Because some patterns are really infrequent the minimum thresholds are very small. That's why S1 has a minimal support of 10%. This value returns an acceptable size of patterns. Because S1 searches for maximal sequential patterns, a huge amount of irrelevant always-repeating patterns are removed. The minimal length is set to 2. Thus, trivial patterns are not discovered. The maximal length is set to an obvious high value of 100. This threshold is actually not necessary. S2 receives a minimal frequency of 2. This results in many subgraphs, however some reference patterns have a frequency of 2. The graphs should have at least two nodes and one edge. This allows complex enough graphs with some expressiveness. S3 discovers maximal repeats and has a longer pattern length preference. This ensures more non-trivial patterns. However, every abstraction that contains only one interaction is removed in a postprocessing step. S4 searches for $n \in \{2,3,4\}$ and $k \in \{0,1,2\}$. The n allows the discovery of minimal non-trivial patterns ($n = 2$) as well as more complex tasks ($n = 4$). The zero skips ($k = 0$) let the approach discover simple n-grams. A $k > 0$ introduces skips and tests the acceptance of skips in this approach. For each functional cluster the most frequent k-skip-n-gram is obtained while the others are removed. This reduces the amount of skip-grams and presents only the most frequent ones.

The transaction identification approaches are configured as follows: The first supporting approach doesn't need a parameter. The result is dependent on process and observer events. However, the second supporting approach splits if interactions are remarkable temporally separated. If two interactions are longer then a quarter of an hour temporally distant, they are separated. TI1 doesn't need a parameter. The outcome is depended on the functional interaction occurrence. In case of TI2, the relative frequency of navigation interactions is 40%. This guess reflects that more content elements exist. The average transaction size is similar to the size of the reference patterns. TI3 uses the crawl to find backward references. This approach returns frequently better results then TI2. TI4 has a time window of 120 seconds. This returns similar good results like TI2. The minimal transaction size is 2. This encourages the strategies 1–3 to discover non-trivial patterns.

2.7.5 Pattern Analysis

Finally, five evaluators judge the results of the strategies. The evaluators are the 5 participants occurring in the 25 user-program pair list.

- Participant 1
- Participant 4
- Participant 6
- Participant 7
- Participant 8

Table 13: The 5 evaluators from the 25 user-program pair list

The evaluation setup in the previous section is applied to the 25 user-program pairs. Every participant receives his/her resulting program patterns. The patterns are presented as follows: There is a tab page for each strategy. A tree shows for each transaction identification approach a list of frequency ordered patterns.

For each discovered pattern a pattern score $\in \{-3, -2, -1, 0, 1, 2, 3\}$ has to be selected. It's a seven point Likert-type scale with the following text on the left side: "This recurring interaction pattern describes a task accomplishment". A score of -3 means that the description "doesn't apply" to the selected pattern, while a score of 3 signifies that it "applies". The default selected score is 0. Moreover, the evaluators was asked to add an optional name for that pattern. The following text motivated the evaluators for that name: "How would you name the task accomplishment? Why did you perform these interactions?" The name is encouraged if the score is > 0. After a score on the scale is selected and an optional name is entered, a button saves the input and shows the next pattern of that strategy.

A special selection policy has to be applied: In some cases the evaluators didn't understand the GUI elements. This comes from insufficient labeling by the application softwares. They couldn't reproduce the patterns and were clueless. In this case the evaluator was asked to skip that pattern without selecting a score. Thus, the score 0 is stored. Because the evaluator couldn't give a statement, the score stays neutral.

To reduce the uncertainty the most frequent elements in the patterns are shown before the evaluation. For example, the *Button-2998* is known to be the button which closes an email. The evaluators are asked to open the corresponding application software to reconstruct the interactions. Thus, evaluators could better suggest the semantic of the interactions.

In the case of S1, the sequences are aligned vertical. This eases the reading and prevents scrolling. An increasing number for each interaction make the sequential semantic clear.

While sequences are easy to understand, graphs can be complexer. The most graphs are a succession of connected nodes. However, sometimes branches or cycles exist. In the case of branches the interpretation is a disjunction. Cycles are interpreted as repeating tasks.

The abstractions of S3 are sets, however they are presented in the same way as S1. This causes a score < 0 if the interactions are in wrong order.

In the case of S4, the skip-grams are aligned horizontal. Each row shows another equal determined skip-gram instance. Each column represents a gram. This is clarified with an increasing number in the column header.

During the evaluation comes to light that it's easier for evaluators to judge the patterns by interestingness and meaningfulness. General meaningful patterns receive a score > 0. Patterns representing tasks that are more interesting receive a higher score. In contrast, more meaningless patterns receive a lower score < 0.

The following table and figures demonstrates the outcome of the evaluation. The quantity and pattern score is presented in detail for every user-program pair. Quantities and average pattern scores are analyzed while ignoring the user-program pairs. A distribution of pattern scores is depicted. The comparison of transaction identification approaches and strategies is visualized with error bars. Finally, the average pattern score of all n/k values of strategy 4 are investigated.

Table 14 presents the evaluation for each of the 25 user-program pairs. All four strategies (S) are shown. In case of S1–S3 the outcome of the four transaction identification approaches (TI) is revealed. In case of S4 the outcome of all n/k values is exposed. On the right side the sum of the patterns and the average pattern score for every user-program pair is presented. The first line of each cell reflects the quantity of discovered patterns. The second line shows the average pattern score. If no patterns are discovered, no average pattern score can be calculated (denoted by "—"). Although the distinct functional interaction quantity drops, there are still some patterns discovered. Thus, there seems no proportion between these two values. Some pairs don't have enough interactions. That's why the strategies commonly return zero patterns. In case of S4 a greater n results in fewer patterns. Hence, long equal sequences are rare. The highest average pattern score of 1.74 is from participant 6 using StarMoney. On the second place with 1.44 is participant 1 working with pgAdmin.

Altogether the strategies discovered 1429 patterns. Figure 15 shows the number of discovered patterns in a bar chart. The bar chart shows every strategy with every transaction identification approach as well as n/k value. Finally, the transaction identification approaches are investigated ignoring the strategy. The most patterns are returned by S4. As expected, the 2-gram quantity is greater then the 3-gram quantity, while this quantity is greater then the 4-gram quantity. For each skip size this results in descending stairs. In case of S1 367 patterns are discovered. The transaction identification approaches behave different in diverse strategies. There is no universal statement about the distribution. However, ignoring the strategies TI3 returned the most patterns. This comes from the fact that the maximal forward crawl identifies transactions which don't often separate frequent sequences.

Figure 13 shows the distribution of pattern scores. The x-axis depicts the possible pattern score values. The y-axis shows the quantity of patterns which received that score. 226 patterns are valued 0. This comes from the fact that many patterns weren't understand by the evaluators. Insufficient labeling of GUI elements prevent the comprehension. If a pattern seems acceptable they are rated with a score of 2. Only 104 pattern received a value of 3. In contrast, the negative values give indication that a bad pattern is scored -2 or directly -3. 237 patterns received the score of -3. Hence,

there are many really bad patterns (-3) but few very good patterns (+3). However, 334 are valued 2 which indicates at least meaningful patterns.

Figure 16 illustrates the average pattern score in a bar chart. The bar chart demonstrates every strategy with every transaction identification approach as well as n/k value. Finally, the transaction identification approaches are investigated ignoring the strategy. S1 seems to work best with TI1 because this combination accomplishes an average pattern score of 0.211. This comes from the fact that S1 searches for maximal sequential patterns. Because every transaction of TI1 ends with a functional interaction, the resulting patterns are more accepted. S2 works best with TI2 and receives an average pattern score of 0.791. The frequent subgraph mining is less restrictive then the maximal frequent sequential mining. The cut-off time of TI2 separates the transactions in such a way that acceptable graphs return. S3 receives the highest values of 0.556 with TI4. The time window generates suitable transactions for the discovery of maximal repeats and abstractions. S4 performs very good without skips. However, if skips are introduced the average pattern score drops noticeable. S4 with 4-grams and no skips outperforms S1–S3. However, ignoring the transaction identification approaches, S2 receives the highest value of 0.374. Although S3 returns sets of interactions, it's on the second place with a value of 0.247. The consideration of the transaction identification approaches ignoring the strategies gives no remarkable difference.

Figure 17 shows the four transaction identification approaches (TI) with an error chart. The strategies are ignored in this comparison. The points visualize the average pattern score. The line depicts the standard deviation of the pattern scores. The standard deviation is always nearly 2. The scores indicate that they aren't close to the mean. A wide distribution of both good and bad patterns are returned. Desirable would be a smaller standard deviation to argue an agreement. However, this shows that more insights are necessary to present only good patterns to the evaluators. Because the mean is for all TIs around 0 a comparison is not expressive enough.

Figure 18 shows the four strategies (S) with an error chart. The transaction identification approaches are ignored in this comparison. The points visualize the average pattern score. The line depicts the standard deviation of the pattern scores. The standard deviation is almost always nearly 2. However, S3 has a standard deviation of 1.675. Thus, there is a bit more agreement to the mean of 0.247. While S1 and S4 have negative, S2 and S3 have positive means. But they are all around 0 and never exceed 1. However, S4 would perform better if no skips are used. Considering all patterns without skips (k=0) S4 obtains an average pattern score of 0.640. However, this is not noticeable higher. All in all, no strategy proves to be significant better then the others.

Figure 14 demonstrates the ordered average pattern score of S4 with different n/k values. For each k another color is used. Clearly we see that skips produce accepted patterns. This comes from the fact that the evaluator miss some important interactions in between. For 4-grams with skips the result is the worst. However in case of no skips, the greater the n grows the better the result. This means that more meaningful interaction sequences emerges, the longer the interaction sequences are. However, at some point ($n > 4$) it becomes more difficult to discover equal n-grams. The average pattern score will drop too, because the n-grams are too long.

The following passage analyses the most acceptable patterns. It presents some of the best patterns with a pattern score of 3. Additionally, optional names by the evaluators are revealed.

Figure 12 shows the quantity of the most accepted patterns from all four strategies. The first four bars demonstrates the quantity in case of a pattern score of 3. The next four bars demonstrates the same with a score of 2. From 104 3-score-patterns 42 are discovered by S1. However, S1 also returns many low rated patterns which results in an average pattern score of -0.104. Only 8 abstractions are rated with a score of 3 in S3. In case of 2-score-patterns S4 is the winner. S3 remains below all of them. In average this strategy archives an average pattern score of 0.247. It's because no good enough patterns are discovered.

The following automatically discovered patterns are rated by corresponding evaluators with a score of 3. In total 67 patterns received that score and at the same time a task description. An extract of 10 patterns are demonstrated below.

The patterns are presented as follows: The description is the optional name translated in English. Some evaluators wrote more then just some keywords. That's why it is more a description then a specific name. The context is the application software where the pattern was discovered. The type of the pattern is one of the following: Sequential, Graph, Abstraction or N-Gram. Finally, the pattern is visualized.

Description Print certain pages of a document
Context Microsoft Word
Type Sequential
Pattern
1. Tab-120944 Registerkarten des Menübands,
2. Button-141909 Registerkarte "Datei",
3. Edit-141955 Seiten: Geben Sie Seitenzahlen und/oder Seitenbereiche. . . ,
4. Button-141944 Drucken

Description Search text snippet from clipboard
Context Adobe Reader
Type N-Gram
Pattern

1. Mouse Left Click Edit-96608 Suchen,
2. Keyboard Control + V

Description Create a new picture and don't save previous modifications
Context Microsoft Paint
Type Sequential
Pattern

1. MenuItem-199798 Neu Ein neues Bild erstellen,
2. Button-199819 Nicht speichern

Description Create a new picture with the menu
Context Microsoft Paint
Type Abstraction
Pattern

- Mouse Left Click Button-140161 Anwendungsmenü,
- Mouse Left Click MenuItem-199798 Neu Ein neues Bild erstellen

Description Save and close the window
Context Microsoft Paint
Type Sequential
Pattern

1. Button-199828 Speichern,
2. Button-199580 Schließen Schließt das Fenster

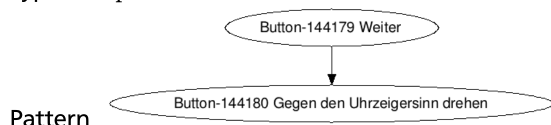
Description Show specific datasets
Context pgAdmin III
Type N-Gram
Pattern

1. Mouse Left Click TreeItem-51327 cluster_s. . . ,
2. Mouse Left Click TreeItem-51328 cluster_s. . . ,
3. Mouse Left Click TreeItem-51327 cluster_s. . . ,
4. Mouse Left Click MenuItem-51351 Daten anzeigen

Description Sign and send email
Context Microsoft Outlook
Type Sequential
Pattern

1. Tab-39230 Registerkarten des Menübands,
2. Button-104386 Signieren,
3. Button-104282 Senden

Description Turn picture right around
Context Windows Photo Viewer
Type Graph
Pattern



Description Respond to email
Context Microsoft Outlook
Type Sequential
Pattern
1. Button-39269 Antworten,
2. Button-104402 Senden

Description Search
Context Notepad++
Type N-Gram
Pattern
1. Mouse Left Click Tab-38729 Tab,
2. Keyboard Control + F Edit-126024 Suchen nach

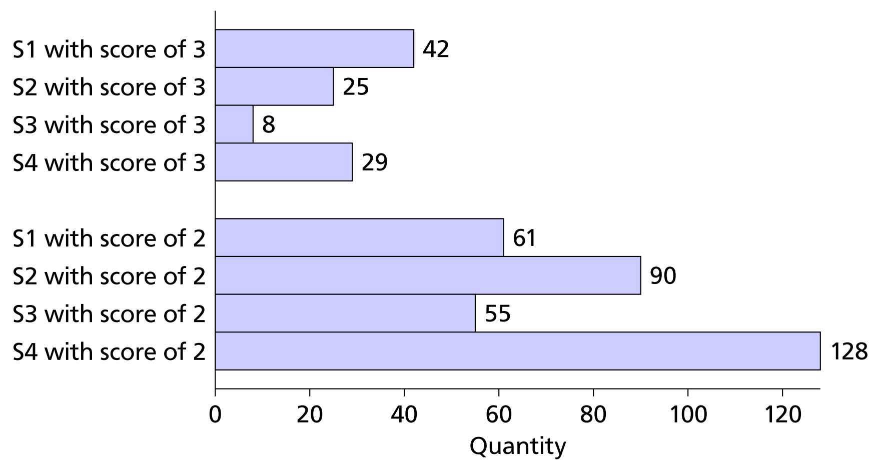


Figure 12: Ordered pattern score quantity of all strategies regarding to 3 or 2 rated patterns

Part.	App.	S1				S2				S3				S4									Sum
		TI1	TI2	TI3	TI4	TI1	TI2	TI3	TI4	TI1	TI2	TI3	TI4	n=2 k=0	n=2 k=1	n=2 k=2	n=3 k=0	n=3 k=1	n=3 k=2	n=4 k=0	n=4 k=1	n=4 k=2	Avg.
8	javaw	2 -1.00	4 0.00	9 0.33	2 -1.00	14 -1.50	12 0.50	10 -0.30	10 -0.90	11 -0.18	10 0.50	17 0.06	5 0.80	19 0.21	13 0.31	9 0.33	10 0.90	7 0.43	4 1.00	5 1.60	2 -2.00	0 —	175 0.06
8	outlook	3 0.67	1 0.00	4 0.75	0 —	23 -1.43	15 -0.47	29 -1.17	5 -0.80	17 -1.59	11 -0.27	21 -0.90	1 -1.00	27 -0.22	25 -1.00	17 -1.06	11 0.64	7 0.43	2 -1.00	2 2.00	1 1.00	0 —	222 -0.72
6	thunderbird	4 0.00	4 2.50	4 2.50	4 3.00	44 1.11	25 1.68	28 1.36	8 1.00	21 0.86	15 0.80	17 1.53	4 1.00	17 0.18	16 -2.31	20 -1.25	16 1.25	19 -2.05	18 -1.56	11 1.91	18 -2.28	17 -2.06	330 0.21
1	winword	2 3.00	5 0.00	6 0.50	5 -1.20	0 —	1 1.00	1 2.00	0 —	0 —	1 1.00	1 2.00	0 —	2 1.50	1 2.00	1 0.00	1 2.00	0 —	0 —	0 —	0 —	0 —	27 0.59
8	winword	2 0.00	2 0.00	0 —	0 —	0 —	0 —	0 —	0 —	1 0.00	1 0.00	0 —	0 —	1 1.00	0 —	1 0.00	0 —	0 —	0 —	0 —	0 —	0 —	8 0.13
1	notepad++	2 -3.00	4 0.50	2 -2.50	4 -1.75	2 -0.50	3 0.67	1 -3.00	0 —	1 3.00	2 0.00	0 —	2 0.00	3 -1.00	2 -0.50	2 -3.00	0 —	0 —	0 —	0 —	0 —	0 —	30 -0.83
1	pgadmin3	1 3.00	8 -2.25	2 3.00	4 2.50	1 3.00	0 —	4 2.75	3 2.33	1 -3.00	0 —	3 -1.33	3 0.67	3 2.33	3 3.00	3 3.00	3 3.00	3 3.00	0 —	3 3.00	0 —	0 —	48 1.44
7	winword	1 2.00	7 0.00	7 0.29	5 0.20	1 2.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	21 0.33
1	texstudio	2 -3.00	3 -3.00	0 —	3 -3.00	0 —	0 —	0 —	0 —	0 —	1 0.00	0 —	0 —	3 1.33	4 -2.00	3 -1.67	1 2.00	1 1.00	1 -3.00	0 —	0 —	0 —	22 -1.50
4	dllhost	1 3.00	1 3.00	6 1.00	1 3.00	1 3.00	0 —	0 —	1 3.00	0 —	3 0.67	0 —	3 1.67	4 0.75	5 2.00	5 1.00	2 1.50	4 2.25	3 0.33	1 2.00	3 0.33	3 1.00	47 1.38
1	acrord32	3 0.67	5 -1.40	0 —	3 -3.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	1 3.00	0 —	1 3.00	0 —	0 —	0 —	0 —	0 —	0 —	13 -0.62
6	starmoney	9 1.33	2 0.50	9 0.33	4 0.50	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	24 0.75
6	starmoney	2 1.00	3 -0.67	1 0.00	1 -2.00	9 1.00	3 0.33	3 0.67	6 0.17	5 0.80	6 0.00	3 0.67	4 0.00	3 0.67	4 -0.25	6 -0.17	4 -0.25	5 0.60	4 0.00	3 -0.67	1 1.00	2 0.00	78 0.23
6	starmoney	2 0.00	4 2.00	6 -0.83	6 0.67	2 1.00	1 2.00	1 -2.00	1 2.00	1 -2.00	1 2.00	0 —	1 1.00	2 0.00	2 2.00	0 —	1 2.00	0 —	0 —	0 —	0 —	0 —	31 0.58
6	winwein	4 0.75	5 0.40	9 0.00	8 -0.38	0 —	0 —	1 0.00	0 —	0 —	1 0.00	0 —	1 0.00	1 0.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	30 0.07
8	powerpnt	0 —	1 -2.00	1 -2.00	1 0.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	3 -1.33
6	winwein	4 0.75	8 0.00	6 0.00	9 -0.33	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	1 0.00	2 -3.00	0 —	0 —	0 —	0 —	0 —	0 —	30 -0.20
8	keepass	3 -3.00	7 -2.43	7 -2.43	7 -2.43	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	24 -2.50
8	pdfxcview	3 -3.00	3 -3.00	3 -3.00	5 -3.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	0 —	14 -3.00
4	mspaint	3 0.00	4 2.75	9 0.56	8 -0.38	0 —	1 3.00	1 3.00	0 —	0 —	1 3.00	1 3.00	0 —	1 3.00	1 -2.00	0 —	0 —	0 —	0 —	0 —	0 —	0 —	30 0.87
1	outlook	2 1.00	6 0.00	7 -0.29	9 -0.56	0 —	1 2.00	0 —	1 2.00	1 2.00	1 2.00	0 —	0 —	1 2.00	0 —	2 -2.50	0 —	0 —	0 —	0 —	0 —	0 —	31 0.00
8	calc	0 —	0 —	1 -3.00	4 0.25	0 —	0 —	0 —	1 -3.00	0 —	0 —	0 —	1 -3.00	1 -3.00	1 -3.00	1 -3.00	0 —	0 —	1 -3.00	0 —	0 —	0 —	11 -1.82
7	acrobat	0 —	3 0.67	3 0.67	1 2.00	8 1.00	3 -0.67	6 1.33	1 2.00	3 -0.67	4 0.00	5 1.20	1 2.00	4 1.00	9 -0.11	8 -0.13	7 -1.57	12 -1.83	7 -0.71	6 -1.00	8 -1.63	6 -1.00	105 -0.31
7	dllhost	0 —	6 0.50	4 0.75	8 1.13	0 —	1 1.00	0 —	0 —	0 —	1 1.00	0 —	1 1.00	1 1.00	5 1.60	2 1.50	2 1.00	3 2.00	2 2.00	1 1.00	3 2.00	0 —	40 1.23
6	starmoney	2 2.00	3 0.33	1 2.00	2 1.50	2 2.00	1 2.00	1 2.00	0 —	2 2.00	0 —	1 2.00	0 —	3 2.00	2 2.00	2 2.00	3 1.33	2 2.00	3 2.00	2 2.00	1 2.00	2 1.50	35 1.74

Table 14: The 25 user-program pairs with regard to the evaluation results for each strategy (S), transaction identification approach (TI) and n/k value

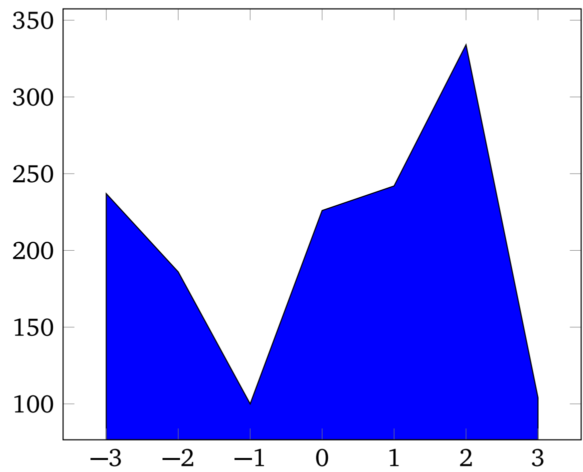


Figure 13: Pattern score chart which shows the quantity of scores

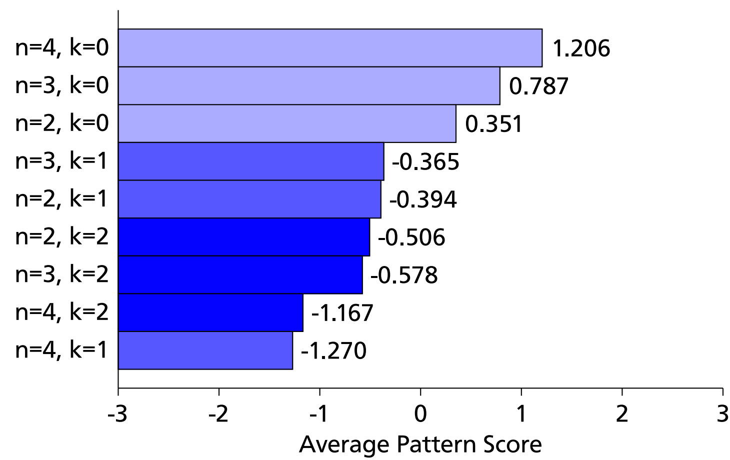


Figure 14: Ordered average pattern score depending on n and k parameter of the k-skip-n-gram strategy (S4)

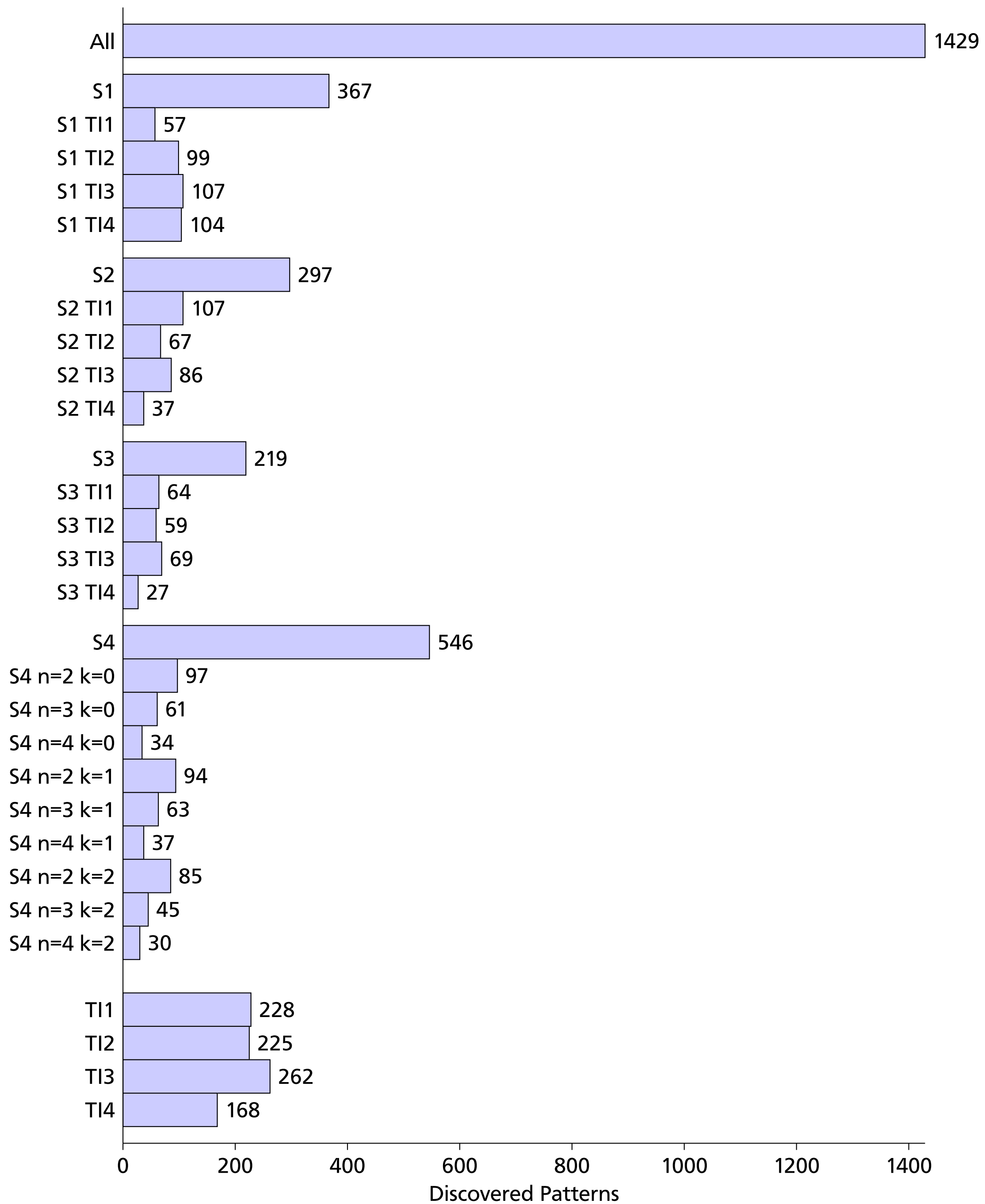


Figure 15: Quantities of discovered patterns depending on different strategies (S) and transaction identification approaches (TI)

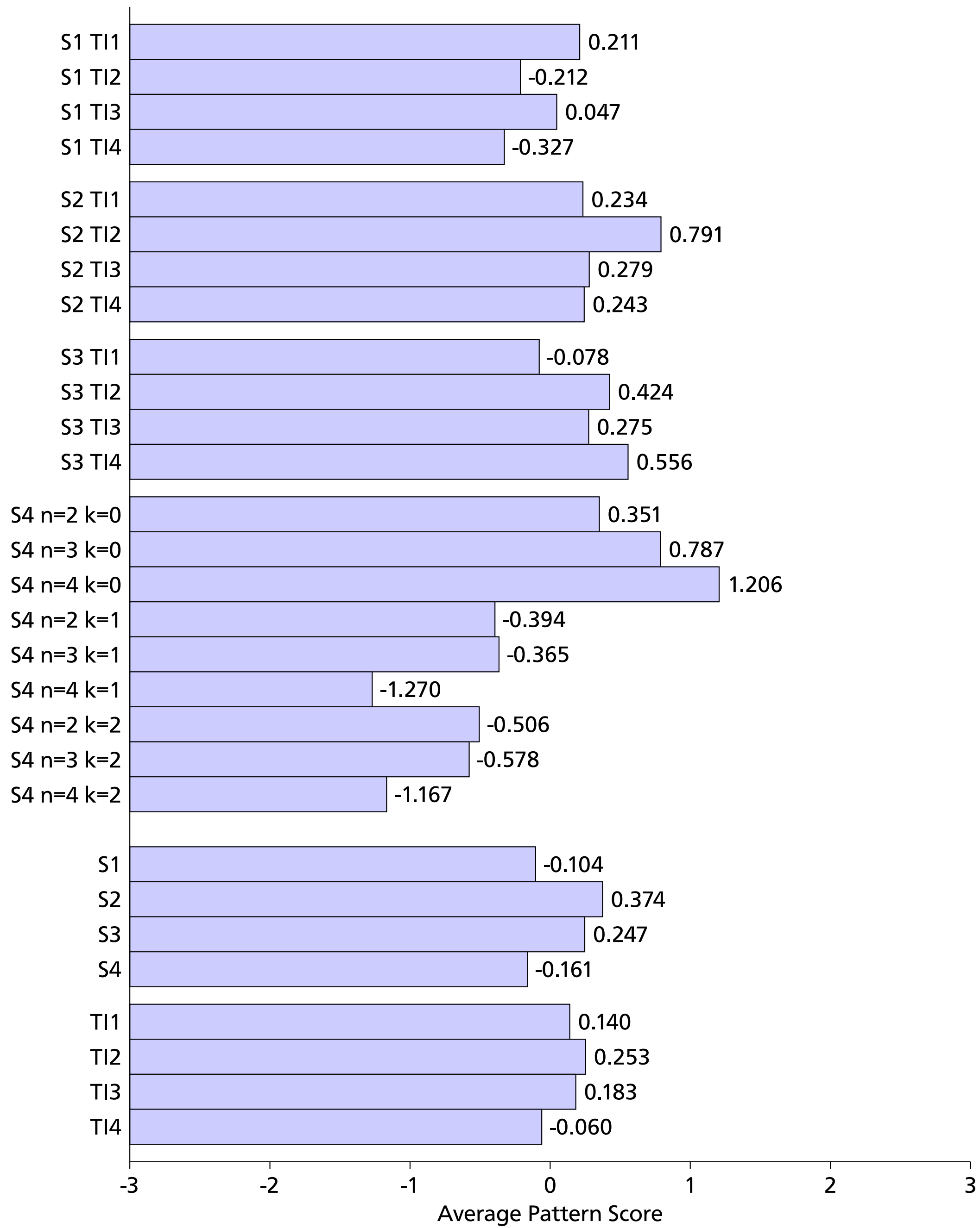


Figure 16: Average pattern score depending on different strategies (S) and transaction identification approaches (TI) as well as n/k values

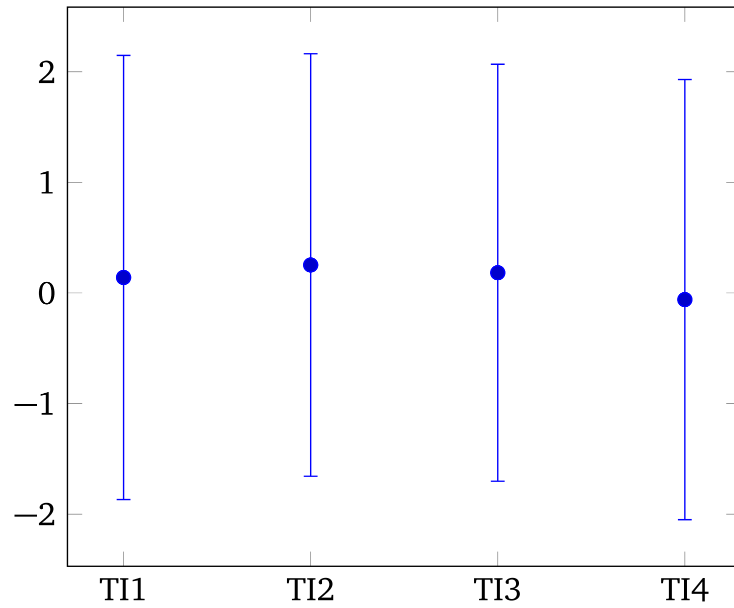


Figure 17: Comparison of the four transaction identification approaches with an error chart ignoring the strategy

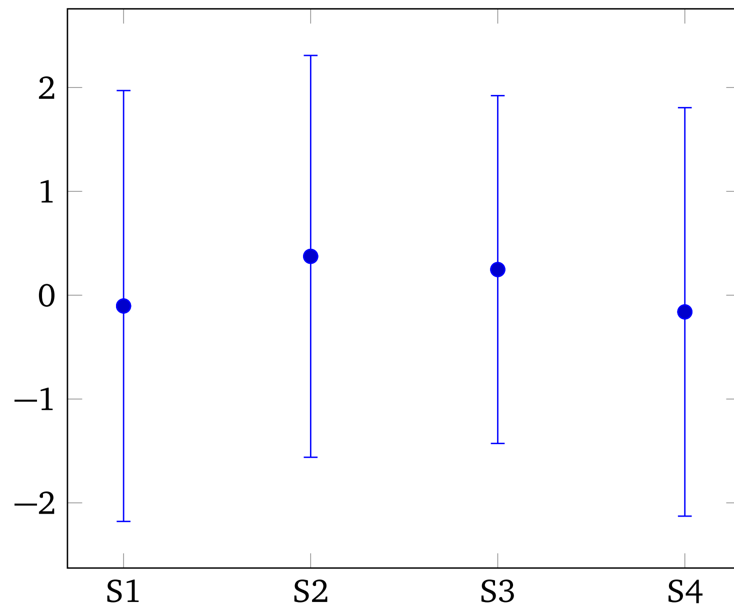


Figure 18: Comparison of the four strategies with an error chart ignoring the transaction identification approaches and n/k values

2.8 Conclusion

The conclusion summarizes the thoughts and discusses the approach. Finally, an outlook argues future promising research achievements.

2.8.1 Summary

The introduction to the topic was about new insights in the usage of interfaces. These thoughts was bridged in the context of the GUI. The definition of meaningful interaction sequences, so called interaction patterns, were proposed. The central question asked about the possibility of discovering these patterns.

The existence of patterns were argued with the concept of intentional action. The question *why* exposed that tasks can be expressed in a sequence of interactions. Thus, patterns were defined to be reoccurring sequences of interactions which express intentional actions. The main problem is the discovery of these patterns in the given interaction log from the first part. This approach is labeled Desktop Usage Mining and is specified to GUI Usage Mining.

Other domains already defined the concept "pattern". However, patterns are frequently used as a building tool, so called pattern languages. That's why it was argued that patterns exist already in online help. A simple example illustrated that a task description is in fact a sequence of interactions. Web Mining is one of the fields of research that wants to discover patterns, too. Web Usage Mining gives insights how they archive it. This translates to Frequent Pattern Mining and to two promising mining strategies: Sequential Pattern Mining and Frequent Subgraph Mining. Additionally, Process Mining promised similar challenges and solutions. The definition of maximal repeats and abstractions fitted to the pattern definition. That's why Process Mining became the third pattern mining strategy.

The related work focused Web Mining in practice. This introduced how Web Usage Mining discovers patterns from click streams. The web log can be compared with the interaction log. Thus, it was clear that similar approaches are possible. The last Web Mining paper gave an outlook what could be possible if patterns are discovered. The paper suggested recommendation based on user profiles. The related Graph Mining paper demonstrated the conversion of access histories to graphs. This idea was adopted to the interaction log. Two Process Mining illustrated the usage of ProM — a framework which consists of Process Mining algorithms. The first paper presented a two-phase approach to discover process maps based on abstractions. The second paper had similar usage data from a touristic system and discovered useful process models. The last paper used Bayesian user models to predict help topics. This gave a motivation what could be possible with the discovered patterns.

The approach consisted of the annotation of reference patterns, the preprocessing of the interaction log and four strategy applications. A first idea how patterns look in real world gave the annotated reference patterns. After the investigation of them it was decided how the interaction log is preprocessed. Only the relationship between one user using one application software were considered. The EOI and its generalized form was determined. Every interaction was assigned to one of the seven classes: Structural, Semi-Structural-Informative, Informative, Semi-Informative-Functional, Functional, Semi-Functional-Structural and None. Repeating EOIs in the interaction log were removed. Finally, four transaction identification approaches were applied. A transaction is a meaningful cluster of interactions. The first approach generated transactions based on functional interactions. The second approach used a reference length for the separation decision. The third approach adopted the maximal forward reference idea to crawls and GUI elements. The fourth approach separated the interaction log based on a time window. Minimal and unnecessary transactions were removed in a postprocessing step. The first strategy was Sequential Pattern Mining which discovers maximal sequential patterns from a sequence database. This was implemented with the help of SPMF. The second strategy was Graph Mining which discovers frequent subgraphs from a graph database. ParSeMiS provided the necessary gSpan algorithm for the implementation. The third strategy was Process Mining which discovers abstractions based on maximal repeats. The ProM framework implemented the used Pattern Abstractions package. The fourth strategy was based on frequent k-skip-n-grams. This novel algorithm was implemented to compensate the disadvantages of the previous strategies.

The evaluation analyzed 25 promising user-program pairs based on the distinct functional interaction quantity. The interaction time was investigated. All four transaction identification approaches were highlighted. The focus was the shape of the transactions (separation and size) depending on different parameters. A common pattern mining setup was configured based on discovered reference patterns. This setup was used to discover 1429 patterns from the 25 user-program list with all four strategies. 5 evaluators judged these patterns with a Likert scale. Unfortunately, the results were not significant enough because of insufficient data points. However, 104 patterns are rated with the highest score. Some of them were briefly introduced. In case of S4 the evaluation showed that skips worsen the discovered patterns. The evaluation indicates that there is still room for discovery improvements.

2.8.2 Discussion

The following passages discuss the approach. Improvements are suggested for future research. A lesson concludes the discussion.

Patterns seem to be a good abstraction. However, there are too complex because of there sequential and time ordered aspects. Simpler associations among GUI elements could ease the discovery. These insights could be similar expressive. However, the nature of task accomplishment in the GUI is a sequence of certain shortcuts and clicks.

The reference patterns were a good starting point to conceptualize patterns. However, a longer study on reference patterns would improve further design decisions. 11 reference patterns of 3 annotators can never show all aspects of patterns. But they helped noticeable to preprocess the interaction log.

The whole part focused on the 1:1 relationship between users and application softwares. But n:1 relationships are more interesting. The merging of different versions could give more data points and more expressive statements. If many users perform the same pattern, the pattern becomes more interesting. Unfortunately, the merging of versions was not considered in the first part.

The generalized EOI proved to be a good idea. The generalization enables the matching of too specific GUI elements. However, the generalization can worsen the matching if errors made. Not every control type and thus possibility is considered in this approach. For example, in (3) DataItem elements also appear in DataGrid elements. That's why some generalizations return no elements (null values). This could alienate the generalized interaction log.

The classification of interactions turned out to be helpful. However, only two control types are functional: Buttons and MenuItems. In the case of MenuItems, only leafs have the same behavior as buttons. There is no distinction in this fact. Furthermore, Semi-Informative-Functional elements also have a functional aspect. They are alterable and can invoke functionality, too. For example, an Edit (input field) starts automatically a search routine if the user enters text. No button is involved that would signalize a functionality. Hence, a more fine-grain classification would be desirable.

The removal of repeating EOIs don't consider different event types. For example, a shortcut and a click to the same element would result in two consecutive interactions with the same element. The algorithm would remove the second click. This attempt is helpful for reducing unnecessary repetitions, however it's still too simple and alienate the interaction log, too.

Four transaction identification approaches are proposed. Three of them have a scientific background in Web Usage Mining. The first approach is intuitive and based on observation. A transaction is a meaningful cluster of interactions. However, a pattern is defined similarly. That's why discovering meaningful segments is actually the same as discovering patterns. But it is difficult to determine the beginning and ending of a task. The best discovery algorithm fails if the input transactions are malformed. That's why more thoughts have to be made here. Because the transaction identification is so important additional evaluations should have been made. Similarly to reference patterns, reference transactions should have been annotated by participants. This would give more clues for the identification.

S1 applied maximal frequent sequential pattern mining on the transactions. The transformation and application was easy. However, the sequence database only consists of generalized EOI IDs. The additional interaction data wasn't used. The average score reveals a mix of unacceptable and acceptable patterns.

S2 applied frequent subgraph mining on the transactions. Element graphs can have the same form as sequences. However, graphs allow the modeling of complex branches and loops. But this complexity was rarely used. That's why it produces same patterns as S2. However, the evaluation indicates that this strategy works best with TI2. Additionally, the filtering was less restrictive. Thus, more acceptable patterns could be presented.

S3 applied the discovery of maximal repeats and abstractions. Although abstractions are sets, the average pattern score is close by the others. However, the abstractions also generate a mix of unacceptable and acceptable patterns. Usually, Process Mining is used for a process model generation. A process model could represent the working with the application software. However, this would be a model and not a set of individual patterns. The analysis of the model could give new insights that were hidden first.

To overcome the problem of transaction identification, S4 was proposed. This strategy isn't based on transactions and adds the uncertainty about beginning and ending of tasks in the approach. Additionally, skips are introduced to enable the discovery of slightly different skip-grams. However, the evaluation revealed noticeable, that skips only worsen the result. In contrast, n-grams (without skips) returned in average positive pattern scores. In fact, 4-grams accomplished an average pattern score of 1.206, which is the highest of all.

It has to be admitted that far too few data points are used in the evaluation. Moreover, the interaction log is still too noisy. Thus, no significance could be argued. That's why only some indications are possible. In the end, no strategy returned only acceptable patterns. Thus, the recall was high but the precision was low. However, S4 performed the best if skips aren't used. At least, 104 patterns of the 1429 discovered patterns received a pattern score of 3. Thus, they should be recurring interaction patterns which describe a task accomplishment.

The bottom line is that patterns exists and 104 very acceptable could be discovered. However, the best strategy for the GUI domain is still unclear because of insufficient data points. Still many design decisions can improve the result. However, the first steps have been taken in discovering interaction patterns in a GUI interaction log.

2.8.3 Outlook

This section given an outlook what could be possible in the future with the new insights. Besides some applications, patterns could be used for recommendation in the context of GUI assistance.

Todays Software functionality is deployed in high-functionality applications. IDEs are a good example for application softwares with many possible tasks. The user is overrun with a broad set of visible GUI elements. Some application softwares support the customization of the surface, however only few uses this feature in reality. That's why an automated customization could be a benefit for the user. Two approaches are considered in this context: (1) GUI hiding and in contrast (2) GUI highlighting.

(1) GUI hiding is the automated concealment of unnecessary GUI elements. Based on the patterns one can suggest the necessary elements. The first part evaluation figures out that only a small set of GUI elements are important for the user. With the help of the pattern frequency the approach could detect what are the most important ones. Every irrelevant GUI element only disturb the workflow and task accomplishment. This is apparent if one looks at menu items of menus. A long list of menu items prevent the search of one desired. GUI hiding could draw a black rectangle on irrelevant determined GUI elements. Thus, only possible relevant GUI elements attract attention.

(2) In contrast, GUI highlighting emphasizes relevant GUI elements. Because the patterns have a sequential aspect the next GUI element can be predicted. If a current EOI occurs in a pattern, this pattern could be performed by the user. GUI highlighting will lead the user to the next meaningful GUI element. This becomes a benefit if foreign patterns are used. Power user have to work with application softwares in deep. That's why they produce more interesting patterns. Normal or weak users could benefit from these patterns. The benefit comes from the fact that GUI highlighting emphasizes the next GUI element based on observation from more experienced power user.

Both approaches become better, the more observation is done. The more observation, the more accurate patterns. It is like a learning algorithm which learns the frequent clicks and keystrokes from the user. Thus, one can claim that patterns can be used to build user models.

The most promising benefit is in the context of predictive GUI assistance. The predictive power of patterns could be utilized for recommendation. The current context of the user makes it possible to select only few relevant patterns. A list of recommended patterns reflect possible task accomplishments. Based on the acceptance of users the recommended patterns could be utilized as follows:

The user can assign a shortcut to a detected pattern. The longer the pattern is, the more tedious becomes its accomplishment. If the user performs the shortcut, the pattern is performed (e.g. clicked) automatically. This saves time and is a simplification.

This is similar to macro recording. An application visualizes GUI elements that can be used (by drag&drop) to build macros. This manual effort could be supported by patterns. Patterns would be a guess of possible macros. Some editing could extend or reduce patterns and give additional meta-data (like names or ratings).

Another application could be a magnetic mouse pointer. The cursor moves slowly to the next predicted click. A more possible pattern results in a faster or persistent moving cursor. This hint helps users finding the next click based on observed patterns. However, it is questionable if this intervention is desired.

Finally, the most benefit application would be a full automatic pattern execution. Based on the current context suitable patterns are selected. Is the possibility, that the pattern is currently executed by the user, over a specified threshold, would the pattern be executed automatically. A lot of observation would be necessary to have a broad range of patterns. It indicates that the work with the GUI is only a sequence of patterns, which are again a sequence of interactions.

GUI Usage Mining provides new ways in the direction of GUI assistance. First steps have been taken and the beginning promises a lot of useful applications. Hopefully, GUI Usage Mining can be built on in the future.

Conclusion

This thesis entered unknown territory in the field of GUI assistance.

The first challenge was the collection of an interaction log. To the best of my knowledge there is no scientific work which collects a similar fine-grain log. That's why a huge effort was undertaken to receive a sufficiently complete and accurate interaction log. However, investigations showed that still noise exists in the data. That's because it's a time-critical real-time observation in an asynchronous system. Additionally, the Accessibility technology leaves more accuracy to be desired. However, what has been achieved is a detailed interaction log with 17759 desktop interactions. Every click and every shortcut on GUI element level is addressed in depth. But the study of 9 participants is very small, which results in insufficient data points. Nevertheless, the interaction log shows that it is possible to observer users in their day-to-day work on the Desktop. Besides, the capturing of that data is also possible, however turns out to be a very challenging task. This came from the fact that the volatile GUI system had to be made persistent. In doing so, first attempts result in a too strict matching. This incorporates redundant GUI elements and application softwares. Thus, noise was propagated. Even the GUI element under the cursor could not accurately determined. Hence, uncertainty had to be implemented in the data scheme. All in all, graphical software mining turns out to be difficult to implement. Many design decisions are open and can increase completeness and correctness.

The aim of the study in the first part was to gather an interaction log. This data was used to discover patterns. However, first it was doubted if patterns exist in the interaction log. The interactions need a certain homogeneity to reveal patterns based on frequency. It might be the case that users don't produce patterns. Thus, they could act very random and interact with no specific style. However, the investigations revealed that the GUI in many cases forces users to do exactly specific sequences of interactions. This was exemplified with the reference patterns. Humans could annotate them, thus the next question was; can algorithms find them automatically? This question was answered with Web Usage Mining. This field of research already discovered patterns in web log data. The server log is similar to the interaction log. Thus, this approach is not far fetched. However, it was unknown which strategy could find the best patterns. Due to unknown results, four strategies were utilized. The question, if patterns could be discovered, can be affirmed. It was now important to discover interesting patterns.

The outcome of the strategies stands and falls with the shape of the transactions. Because it was unknown, which transaction identification approach works best with the interaction log data, four approaches were applied. Again Web Usage Mining was able to point the way. However, the interaction log seems to be too fine-grain. It turns out that the identification of transactions are the first important step to discover patterns. A transaction is ideally a pattern instance. The strategies only determine frequent generalized transactions and call them patterns. This means that more research have to be made in identification of transactions.

To ignore the transactions, a fourth strategy was invented. While the first three strategies revealed at the beginning some disadvantages, the fourth strategy addressed certain problems. The first problem was the identification of transactions. The n-gram based approach doesn't need transactions. In fact, it tries to discover transactions in form of n-grams. The variable size of transactions is modeled with the n parameter. Similar to the first two strategies, equal n-grams can be collected which determines a frequency. A second problem was the transparency of the algorithms. Sequential pattern mining allows missing items in the sequences and the abstraction in process mining forms more abstract sets. The evaluation revealed that this is not preferable. Thus, n-grams (without skips) have to match exactly regarding to functional interactions. However, skips are introduced to generalize sequences. But the evaluation teaches us that exact sequences should be preferred.

The question about the best strategy could not be answered well enough. Because of insufficient and noisy initial data, the discovery strategies performed with a mix of good and bad patterns. Many of the results are no patterns at all. However, the pattern analysis revealed that 438 patterns are rated with a pattern score of 2 or 3. It is possible that one strategy alone can never return only adequate patterns. A composition of the results could improve the outcome. Before patterns can be applied in a useful application, more work has to be done. The future work should focus on the discovery of only relevant patterns.

The overall conclusion is that the discovery of interaction patterns with GUI Usage Mining is possible, however still too immature.

Glossary

API Application Programming Interface. 7, 9, 18, 33

application software (application, software program, program) is a program designed for the end-user that has a graphical user interface. 2, 4, 6–21, 23–25, 27, 29, 33–41, 43–46, 48–52, 57, 58, 60, 69, 70, 78–81

CLI Command-Line Interface. 37

crawl is a term used to describe the process of a software program that requests another application software to receive the graphical user interface elements. Further the term is used to define a set of GUI elements describing the state of an application software. 7, 9, 39

EOI Element of Interest. 13, 14, 16, 18, 24, 25, 33–35, 43–53, 58, 67, 78–80

gSpan graph-based Substructure pattern mining. 50, 78

GUI Graphical User Interface. 2, 6–11, 13, 15–19, 21, 24–26, 33–42, 44, 47–51, 56, 69, 78–81

ID Identification number. 9, 12, 14, 15, 19, 24, 33, 34, 43, 48–55, 79

IDE Integrated Development Environment. 10, 11, 24, 58, 80

JAR Java Archive. 56

MSAA Microsoft Active Accessibility. 9, 13, 34

OS Operating System. 7, 9, 11, 12, 18, 20, 25

ParSeMiS Parallel and Sequential Graph Mining Suite. 54, 55, 78

PC Personal Computer. 4, 6, 9, 20, 24, 33, 35, 36, 48, 57

PID Process Identification number. 12, 18, 33, 34

ProM Process Mining Framework. 42, 56, 78

SHA Secure Hash Algorithm. 12

SPMF Sequential Pattern Mining Framework. 53, 78

UIA UI Automation. 9, 10, 13, 17, 34, 53

user (end-user) "is any individual who is not involved with supporting or developing a computer or service" [45, (1.)]. 7

VMSP Vertical mining of Maximal Sequential Patterns. 50, 53, 54

WIMP Windows, Icons, Menus and Pointer. 36

XES Extensible Event Stream. 51, 56

XML Extensible Markup Language. 56

XSD XML Schema Definition. 56

References

[1] W.M.P. van der Aalst. *Event logs*. Apr. 16, 2011. URL: http://www.processmining.org/logs/start (visited on 01/01/2015).

[2] W.M.P. van der Aalst and A.J.M.M. Weijters. "Process mining: a research agenda". In: *Computers in Industry* 53.3 (2004), pp. 231–244.

[3] Pekka Aho et al. "Murphy Tools: Utilizing Extracted GUI Models for Industrial Software Testing". In: *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops* (2014).

[4] Christopher Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[5] Christopher Alexander. *The Timeless Way of Building*. Oxford University Press, 1979.

[6] G.E.M. Anscombe. *Intention*. Harvard University Press, Nov. 15, 2000.

[7] Wayne Beaton. *Usage Data Collector (UDC)*. URL: https://wiki.eclipse.org/UDC (visited on 01/01/2015).

[8] Ivan Benc, Mario Stefanec, and Sinisa Srbljic. "Usage Tracking by Public Information System Mediator". In: *Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference* 2 (2004), pp. 723–726.

[9] Paul Blenkhom and Gareth Evans. "Architecture and requirements for a Windows screen reader". In: *IEE Seminar on Speech and Language Processing for Disabled and Elderly People* (2000).

[10] R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. "Abstractions in process mining: A taxonomy of patterns". In: *Business Process Management* (2009).

[11] R.P. Jagadeesh Chandra Bose, Eric H.M.W. Verbeek, and W.M.P. van der Aalst. "Discovering hierarchical process models using prom". In: *IS Olympics: Information Systems in a Diverse World* 107 (2011), pp. 33–48.

[12] Ulrik Brandes, Markus Eiglsperger, and Jürgen Lerner. *GraphML Serialization*. URL: http://graphml.graphdrawing.org/primer/graphml-primer.html (visited on 01/01/2015).

[13] Peter F. Brown et al. "Class-Based n-gram Models of Natural Language". In: *Computational Linguistics* 18 (1992), pp. 467–479.

[14] Pascal Cabanel. *xsd2Code community edition .net class generator from XSD schema*. July 13, 2014. URL: https://xsd2code.codeplex.com/ (visited on 01/01/2015).

[15] John M. Carroll. "Human Computer Interaction - brief intro". In: *Soegaard, Mads and Dam, Rikke Friis (eds.). "The Encyclopedia of Human-Computer Interaction, 2nd Ed.". Aarhus, Denmark: The Interaction Design Foundation* (2014). URL: https://www.interaction-design.org/encyclopedia/human_computer_interaction_hci.html (visited on 01/01/2015).

[16] Luigi Cerulo. "On the Use of Process Trails to Understand Software Development". In: *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06)* (2006), pp. 303–304.

[17] Ming-Syan Chen, Jong Soo Park, and Philip S. Yu. "Data Mining for Path Traversal Patterns in a Web Environment". In: *Proceedings of the 16th ICDCS* (1996), pp. 385–392.

[18] Jan Claes. *Add Artificial Events plugin*. June 23, 2011. URL: http://www.janclaes.info/post.php?post=addartificialevents (visited on 01/01/2015).

[19] Git Community. *git-commit*. URL: http://git-scm.com/docs/git-commit (visited on 01/01/2015).

[20] Git Community. *git-fetch*. URL: http://git-scm.com/docs/git-fetch (visited on 01/01/2015).

[21] Git Community. *git-pull*. URL: http://git-scm.com/docs/git-pull (visited on 01/01/2015).

[22] NetBeans Community. *NetBeans IDE*. URL: https://netbeans.org/ (visited on 01/01/2015).

[23] NetBeans Community. *NetBeans Usage Data Tracking*. URL: http://netbeans.org/about/usage-tracking.html (visited on 01/01/2015).

[24] Wikipedia Community. *Keystroke logging*. URL: http://en.wikipedia.org/wiki/Keystroke_logging (visited on 01/01/2015).

[25] Wikipedia Community. *Software mining*. URL: http://en.wikipedia.org/wiki/Software_mining (visited on 01/01/2015).

[26] "Complete mining of frequent patterns from graphs: Mining graph data". In: *Machine Learning* 50 (2003), pp. 321–354.

[27] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. "Grouping web page references into transactions for mining world wide web browsing patterns". In: *Proceedings 1997 IEEE Knowledge and Data Engineering Exchange Workshop* (1997), pp. 2–9.

[28] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. "Web mining: Information and pattern discovery on the world wide web". In: *Tools with Ariticial Intelligence* (1997), pp. 558–567.

[29] Huizhong Duan, Emre Kıcıman, and ChengXiang Zhai. "Click Patterns: An Empirical Representation of Complex Query Intents Categories and Subject Descriptors". In: *CIKM'12* (2012).

[30] Eclipse Foundation. *Eclipse*. URL: `https://eclipse.org/` (visited on 01/01/2015).

[31] P. Fournier-Viger et al. "SPMF: a Java Open-Source Pattern Mining Library". In: *Journal of Machine Learning Research (JMLR)* 15 (2014), pp. 3389–3393. URL: `http://www.philippe-fournier-viger.com/spmf/`.

[32] Philippe Fournier-Viger. *SPMF Algorithms*. URL: `http://www.philippe-fournier-viger.com/spmf/index.php?link=algorithms.php` (visited on 01/01/2015).

[33] Philippe Fournier-Viger. *SPMF Documentation*. URL: `http://www.philippe-fournier-viger.com/spmf/index.php?link=documentation.php` (visited on 01/01/2015).

[34] Philippe Fournier-Viger et al. "VMSP: Efficient Vertical Mining of Maximal Sequential Patterns". In: *Advances in Artificial Intelligence*. Vol. 8436. 2014, pp. 83–94.

[35] Erich Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Oxford University Press, Nov. 10, 1994.

[36] Christian W. Günther and Eric H.M.W. Verbeek. *XES Standard Definition 2.0*. Mar. 28, 2014. URL: `http://www.xes-standard.org/_media/xes/xesstandarddefinition-2.0.pdf` (visited on 01/01/2015).

[37] David Guthrie et al. "A Closer Look at Skip-gram Modelling". In: *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)* (2006), pp. 1222–1225.

[38] Jonathan Peli de Halleux. *GraphML Serialization*. Jan. 22, 2009. URL: `https://quickgraph.codeplex.com/wikipage?title=GraphML%20Serialization` (visited on 01/01/2015).

[39] Jonathan Peli de Halleux. *QuickGraph, Graph Data Structures And Algorithms for .NET*. Nov. 19, 2011. URL: `https://quickgraph.codeplex.com/` (visited on 01/01/2015).

[40] J. Han et al. "Frequent pattern mining: current status and future directions". In: *Data Mining and Knowledge Discovery* 15 (2007), pp. 55–86.

[41] Rob Haverty. "New Accessibility Model for Microsoft Windows and Cross Platform Development". In: *ACM SIGACCESS Accessibility and Computing* (2005).

[42] Tim Henderson. *The ParSeMiS project*. Mar. 14, 2014. URL: `https://github.com/timtadh/parsemis` (visited on 01/01/2015).

[43] Sven Hertling. "Search Engine for Graphical User Interfaces". MA thesis. Technische Universität Darmstadt, 2015.

[44] Computer Hope. *Process*. URL: `http://www.computerhope.com/jargon/p/process.htm` (visited on 01/01/2015).

[45] Computer Hope. *User*. URL: `http://www.computerhope.com/jargon/u/user.htm` (visited on 01/01/2015).

[46] Eric Horvitz et al. "The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users". In: *Fourteenth Conference on Uncertainty in Artificial Intelligence* (1998), pp. 256–265.

[47] Renáta Iváncsy and István Vajk. "Frequent pattern mining in web log data". In: *Acta Polytechnica Hungarica* 3 (2006), pp. 77–90.

[48] Ni Jin, Wang Mingming, and Wang Jiangqing. "Realization on Intelligent GUI Automation Testing Based-on .NET". In: *Proceedings - 2010 3rd IEEE International Conference on Computer Science and Information Technology, ICCSIT 2010* (2010).

[49] Jozef Kapusta, Michal Munk, and Martin Drlík. "Cut-off Time Calculation for User Session Identification by Reference Length". In: *2012 6th International Conference on Application of Information and Communication Technologies, AICT 2012 - Proceedings* (2012).

[50] Richard Kennard and John Leaney. "An Introduction to Software Mining". In: *SoMeT* (2012).

[51] Xiaosong Li and Rick Mugridge. "Petri Net Based Graphical User Interface Specification Tool". In: *Proceedings Software Education Conference* (1995).

[52] George Mamaladze. *Application and Global Mouse and Keyboard Hooks .Net Libary in C#*. Oct. 9, 2011. URL: `https://globalmousekeyhook.codeplex.com/` (visited on 01/01/2015).

[53] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. “GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing”. In: *Reverse Engineering - Working Conference Proceedings* (2003).

[54] michaelnoonan. *WindowsInput 0.2.0*. Nov. 27, 2013. URL: https://www.nuget.org/packages/WindowsInput/ (visited on 01/01/2015).

[55] Microsoft. *Using Calculator in Windows 7*. URL: http://windows.microsoft.com/en-us/windows7/using-calculator-in-windows-7 (visited on 01/01/2015).

[56] Bamshad Mobasher, Robert Cooley, and Jaideep Srivastava. “Automatic personalization based on Web usage mining”. In: *Communications of the ACM* 43.8 (2000).

[57] Tadao Murata. “Petri Nets: Properties, Analysis and Applications”. In: *Proceedings of the IEEE* 77.4 (1989), pp. 541–580.

[58] Microsoft Developer Network. *Accessibility*. URL: http://msdn.microsoft.com/en-us/library/ms753388%28v=vs.110%29.aspx (visited on 01/01/2015).

[59] Microsoft Developer Network. *AccessibleObjectFromPoint function*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/dd317977%28v=vs.85%29.aspx (visited on 01/01/2015).

[60] Microsoft Developer Network. *Automation Element Property Identifiers*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee684017%28v=vs.85%29.aspx (visited on 01/01/2015).

[61] Microsoft Developer Network. *Computer Names*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms724220%28v=vs.85%29.aspx (visited on 01/01/2015).

[62] Microsoft Developer Network. *Control Type Identifiers*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671198%28v=vs.85%29.aspx (visited on 01/01/2015).

[63] Microsoft Developer Network. *IUIAutomation::AddAutomationEventHandler method*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671508%28v=vs.85%29.aspx (visited on 01/01/2015).

[64] Microsoft Developer Network. *IUIAutomationElementArray interface*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671426%28v=vs.85%29.aspx (visited on 01/01/2015).

[65] Microsoft Developer Network. *IUIAutomation::ElementFromIAccessible method*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671536%28v=vs.85%29.aspx (visited on 01/01/2015).

[66] Microsoft Developer Network. *IUIAutomation::ElementFromPoint method*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ee671538%28v=vs.85%29.aspx (visited on 01/01/2015).

[67] Microsoft Developer Network. *MouseButtons Enumeration*. URL: http://msdn.microsoft.com/en-us/library/system.windows.forms.mousebuttons%28v=vs.110%29.aspx (visited on 01/01/2015).

[68] Microsoft Developer Network. *UI Automation and Microsoft Active Accessibility*. URL: http://msdn.microsoft.com/en-us/library/ms788733%28v=vs.110%29.aspx (visited on 01/01/2015).

[69] Microsoft Developer Network. *UI Automation Control Patterns Overview*. URL: http://msdn.microsoft.com/en-us/library/ms752362%28v=vs.110%29.aspx (visited on 01/01/2015).

[70] Microsoft Developer Network. *UI Automation Control Types Overview*. URL: http://msdn.microsoft.com/en-us/library/ms749005%28v=vs.110%29.aspx (visited on 01/01/2015).

[71] Microsoft Developer Network. *WaitForInputIdle function*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms687022%28v=vs.85%29.aspx (visited on 01/01/2015).

[72] IEEE Task Force on Process Mining. *XES XSD*. Mar. 28, 2014. URL: http://www.xes-standard.org/xes.xsd (visited on 01/01/2015).

[73] John R. Punin, Mukkai S. Krishnamoorthy, and Mohammed J. Zaki. “Web Usage Mining - Languages and Algorithms”. In: *Exploratory Data Analysis* (2003), pp. 1–28.

[74] Vladimir A. Rubin et al. “Process mining can be applied to software too!” In: *ESEM’14* (2014).

[75] Jaideep Srivastava et al. “Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data”. In: *SIGKDD Explorations* 1.2 (2000), pp. 12–23.

[76] GraphML Team. *The GraphML File Format*. Apr. 17, 2013. URL: http://graphml.graphdrawing.org/ (visited on 01/01/2015).

[77] TestStack. *Handling waiting in White*. Sept. 12, 2014. URL: http://docs.teststack.net/White/Advanced%20Topics/Waiting.html (visited on 01/01/2015).

[78] TestStack. *TestStack Documentation*. URL: http://teststack.azurewebsites.net/white/index.html (visited on 01/01/2015).

[79] TestStack. *TestStack.White*. URL: http://teststack.net/White/ (visited on 01/01/2015).

[80] Eric H.M.W. Verbeek. *IEEE CIS Task Force on Process Mining*. Aug. 14, 2013. URL: http://www.win.tue.nl/ieeetfpm/ (visited on 01/01/2015).

[81] Eric H.M.W. Verbeek. *ProM*. Dec. 19, 2014. URL: http://www.processmining.org/prom/start (visited on 01/01/2015).

[82] Eric H.M.W. Verbeek. "ProM6 Getting Started". 2010.

[83] Eric H.M.W. Verbeek. *XES*. Sept. 11, 2012. URL: http://www.xes-standard.org/ (visited on 01/01/2015).

[84] Eric H.M.W. Verbeek and R. P. Jagadeesh Chandra Bose. "ProM6 Tutorial". 2010.

[85] Li Xiong. *MSAA, UIA brief explanation*. Mar. 28, 2009. URL: http://blogs.msdn.com/b/lixiong/archive/2009/03/28/msaa-uia-brief-explanation.aspx (visited on 01/01/2015).

[86] Xifeng Yan and Jiawei Han. "gSpan: Graph-Based Substructure Pattern Mining". In: *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* (2002).