Henri Pitkänen

Exploratory sequential data analysis of user interaction in contemporary BIM applications

Department of Computer Science and Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 24.04.2017

Thesis supervisor:

Prof. Marko Nieminen

Thesis advisors:

M.Sc. (Tech.) Ville Rousu

M.Sc. (Tech.) Osmo Tolvanen

Aalto University
School of Science

AALTO-YLIOPISTO
TIETOTEKNIIKAN LAITOS

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Henri Pitkänen

Työn nimi: Exploratory sequential data analysis of user interaction in contemporary BIM applications

Päivämäärä: 24.04.2017 Kieli: Englanti Sivumäärä: 7+80

Tietotekniikan laitos

Professuuri: Käytettävyys Koodi: T-121

Valvoja: Prof. Marko Nieminen

Ohjaajat: DI Ville Rousu, DI Osmo Tolvanen

Luomiseen perustuva ohjelmisto mahdollistaa käyttäjän työskentelyn oman visionsa ja sääntöjensä mukaisesti. Ohjelmien analysoinnin kannalta tämä on haastavaa, koska ei ole varmuutta siitä, kuinka ohjelmistoa tarkalleen käytetään ja millaisia työskentelytapoja ohjelmiston eri käyttäjäryhmille voi syntyä.

Opinnäytetyön tavoitteena oli tutkia ja identifioida toistuvien käyttäjätapahtumasekvenssien analyysipotentiaalia tietomallinnukseen keskittyvässä luomispohjaisessa ohjelmistossa. Opinnäyte esittelee myös evaluointimallikonseptin, jonka avulla on mahdollista tunnistaa toistuvasta käyttäytymisestä aiheutuvia käytettävyysongelmia. Lopuksi työssä esitellään sekvenssianalyysiin perustuva ohjelmiston käyttäjän toiminta-analyysi sekä ennustava koneoppimisen sovellus.

Opinnäytetyössä esitellään data-analyysisovellus, joka perustuu käytettävyystutkimuksessa esiintyvien toistuvien sekvenssien tai kokeellisesti toistuvien sekvenssien analyysiteorian tarkasteluun. Sovelluksen toteutus on tehty eritoten työssä käytetylle ohjelmistolle, jossa käyttäjän detaljointitapahtumista muodostetaan sekvenssejä sekvenssitietokannan luomiseksi. Työssä käytetään sekvenssien toistuvuusanalyysiin moderneja louhintamenetelmiä nimeltään BIDE ja TKS. Lopuksi työssä hyödynnetään luotua sekvenssitietokantaa myös käyttäjän detaljointityön ennustamista varten käyttämällä CPT+ algoritmia.

Opinnäytetyön tulosten pohjalta pyritään löytämään vaihtoehtoja käytettävyyden ja tuotekehityksen päätöksenteon tietopohjaiseksi tueksi tunnistamalla ja ennustamalla käyttäjien toimintaa ohjelmistossa. Löydetyn informaation avulla on mahdollista ilmaista käytettävyyteen liittyviä ongelmia kvantitatiivisen tiedon valossa.

Avainsanat: Data-analyysi, Koneoppiminen, Käytettävyysanalyysi, Käyttäjäloki, Tiedonlouhinta, Tietomallinnus, Sekvenssi, Toistuvuus

AALTO UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ABSTRACT OF THE
MASTER'S THESIS

Author: Henri Pitkänen

Title: Exploratory sequential data analysis of user interaction in contemporary BIM applications

Date: 24.04.2017 Language: English Number of pages: 7+80

Department of Computer Science and Engineering

Professorship: Usability Code: T-121

Supervisor: Prof. Marko Nieminen

Advisors: M.Sc. (Tech.) Ville Rousu, M.Sc. (Tech.) Osmo Tolvanen

Creation oriented software allows the user to work according to their own vision and rules. From the perspective of software analysis, this is challenging because there is no certainty as to how the users are using the software and what kinds of workflows emerge among different users.

The aim of this thesis was to study and identify the potential of sequential event pattern data extraction analysis from expert field creation oriented software in the field of Building Information Modeling (BIM). The thesis additionally introduces a concept evaluation model for detecting repetition based usability disruption. Finally, the work presents an implementation of sequential pattern mining based user behaviour analysis and machine learning predictive application using state of the art algorithms.

The thesis introduces a data analysis implementation that is built upon inspections of Sequential or Exploratory Sequential Data Analysis (SDA or ESDA) based theory in usability studies. The study implements a test application specific workflow sequence detection and database transfer approach. The paper uses comparative modern mining algorithms known as BIDE and TKS for sequential pattern discovery. Finally, the thesis utilizes the created sequence database to create user detailing workflow predictions using a CPT+ algorithm.

The main contribution of the thesis outcome is to open scalable options for both software usability and product development to automatically recognize and predict usability and workflow related information, deficiencies and repetitive workflow. By doing this, more quantifiable metrics can be revealed in relation to software user interface behavior analytics.

Keywords: AEC, BIM, Event data, Exploratory data analysis, Frequent itemset mining, Machine learning, Repetitive pattern analysis, ESDA, Sequence detection, Sequential pattern mining, Usability Engineering

Preface

This thesis was implemented out of personal interest in finding ways to utilize and learn from user created data. To shed some light on my personal goals, I feel that I have also learned something new.

First, I would simply like to thank everyone at Tekla. I would not want to merely name-drop a few individuals, as I feel that everyone by whom I have been challenged have affected the outcome. However, without Ville Rousu and Jarmo Manninen's extraordinary ability to quickly comprehend and see value, the work would not have been started.

I would like thank the thesis supervisor Marko Nieminen for his enthusiasm, guidance and expert knowledge throughout the project. I would also like to thank the advisor Osmo Tolvanen for his support and ability to see my work from the perspective of usability. I am also grateful to Sirpa Riihiaho for the guidance at the beginning of the work.

Finally, I would like to thank Henri Aalto and Iikka Olli for their interest and wits towards the research topic. Thanks are due to the whole UX-team for the additional support and interest.

Helsinki, 24.04.2017

Henri Pitkänen

Contents

Commonly Used Abbreviations

BIDE	BI-Directional Extension
BIM	Building Information Modeling
CPT	Compact Prediction Tree
ESDA	Exploratory Sequential Data Analysis
HCI	Human Computer Interaction
LSA	Lag Sequential Analysis
MRP	Maximal Repeating Pattern
SDB	Sequence Database
TKS	Top-K Sequential Pattern

1 Introduction

Software usability design is greatly influenced by the main objectives and complexity of the initial *Human Computer Interaction* (HCI) task assigned. Understanding user workflow and the field of work in question is often crucial in order to make the right decision among various trade-offs in usability design (Nielsen, 1993, p. 40–42). In ideal cases, information is gathered by implementing various carefully observed *usability tests* in controlled conditions (Nielsen, 1993; Rubin, 1994, p. 21–26; 112–113). However, the approach might be problematic in cases where the software contains a variety of tools[1] and features that have previously not been evaluated by usability experts. In these situations, the overall software design may become balancing between feature rich content and usability.

When using software, the majority of the users' choices are directly linked to user interface events. Creating a record of such event traces to a log file that contains user event data in an organized format enables user event based behavioral analytics and statistics (Nielsen, 1993; Hilbert and Redmiles, 2000, p. 216–27; 387). In usability analysis, the principle of this kind of data structure is more explicitly defined as *User-Interface* UI event that includes either a *mouse press* or *key press*, an *Identifier* (ID) that refers to the text field or button in the UI, a *timestamp*, and other information, such as input method variations (Hilbert and Redmiles, 2000, p. 387). Because of the structured and sequential nature of the data, it is certainly an area of interest in many studies. In usability design, there have previously been multiple techniques that have automation potential. The first collections of potential techniques were introduced to the public audience in Hilbert and Redmiles (2000, p. 385–387), where eight high level categories were characterized and explained as a guide for understanding the existing research. Another similar collection was by Ivory and Hearst (2001), where multiple existing automated analysis were identified and categorized.

Despite the previous interest in automated usability analysis, there seems to be a limited amount of recent research regarding pattern recognition and analysis among *creation oriented* or *expert software systems*[2]. In spite of a thorough search, no similar research appears to have been done, where usability study driven workflow sequence detection and sequential pattern recognition is created and analyzed using pre-existing large data-sets from Building Information Modeling (BIM) software domain.

The aim of this thesis was to study and identify the possibility to extract user workflow related sequential event patterns from expert field creation oriented software in the field of BIM. The thesis briefly discusses the existing traditional usability approaches that deal with usability disruption in software systems. The paper then

[1] 'Tools' refer to the tools inside the software that help task completion with partial automation

[2] Based on searches in databases such as Association for Computing Machinery (ACM) digital library (dl.acm.org), Elsevier ScienceDirect (sciencedirect.com), IEEE Software (ieeexplore.ieee.org) and Google Scholar (scholar.google.com)

continues to discuss automated pattern recognition as a solution in creation oriented software. This includes an inspection and selection of different types of existing automated *Sequential* or *Exploratory Sequential Data Analysis* (SDA or ESDA) usability analysis theorized techniques that seem feasible in the BIM domain. Additionally, the techniques are viewed and compared from the perspective of modern mining approaches to find modern applications of selected preliminary techniques.

After the theoretical background, the thesis explores the chosen methods. The selected analysis techniques were used as a basis for preliminary usability disruption cases and an approach model for method implementation. Disruption case examples were created for initial categorization purposes. After creating the approach model, the selected techniques were tested using existing test application data. As a first test implementation, a sequence detection algorithm was created to suit the test application data. As a second implementation, the detected user sequences were mined for repetitive patterns using pre-existing open source *Frequent itemset* and *Sequential pattern* mining algorithms library. Additionally, a pattern prediction approach was used to predict from captured user input sequences. This enabled the use of automated pattern discovery and prediction in order to reveal and predict common user behavior and repetitive tasks among captured sequences throughout the test application.

The study has a qualitative basis and its paradigm primarily deals with usability analysis. As the thesis includes the creation of a sequence detection algorithm and mining implementations suitable for pre-existing user event data, it also contains quantitative elements. However, as the implementations and the selection of the most suitable methods are based on inspection, the interpretation of the results remains under the qualitative domain.

The inspiration for the thesis arises from a personal interest in finding new useful information from existing user created data. The study aims to help both usability and software development to automatically recognize relevant deficiencies and introduce more quantifiable metrics in order to understand user needs and behavior in BIM domain. The thesis is carried out as master's thesis for the Trimble Solutions Corporation Structures division that is responsible for Tekla Structures Software. Any applied implementation of the automated analysis is created for existing Tekla Structures data only. Introduction of the test application can be found in section 4.1.

The author is an employee of Trimble Corporation as a Software Specialist in the Tekla Software Product Offering unit. The thesis will be evaluated at Aalto University. The supervisors on behalf of Trimble are M.Sc. Osmo Tolvanen and M.Sc. Ville Rousu. The examiner of the thesis is Prof. Marko Nieminen from Aalto University.

1.1 Justification from value standpoint

Successfully entering the field of expert software systems requires a vast amount of knowledge and a careful interpretation of given requirements. In order to create critical content to have more acceptable software and a wider user-base, one has to choose the correct path from the subjective and proportionally uneven surroundings that contain endless choices. Businesses that do succeed in these fields have created software that serves customer goals, is stable and allows experts to achieve their goals.

As the product market has matured over time, there has been an increase in acceptability requirements and especially in the role of usability due to its significance in increased loyalty and user satisfaction (Grudin, 1991; Rajanen and Iivari, 2007, p. 187–191; 511–512). Utilising usability activities to identify the most critical problems in expert systems can be difficult because of the expert field knowledge requirements. Additionally, the user interface can be costly and great in depth if analyzed thoroughly using traditional[3] usability evaluation methods.

Many of the previous studies are based on defining the cost-benefit structures of usability design. Definitions are built upon different viewpoints, commonly either company, customer, or the end user on some level. Ehrlich and Rohn provide a classification of usability benefits divided into three different areas: *Increased sales*, *Reduced support costs*[4] and *Reduced development costs*. Ehrlich et al. have also presented some case studies linking these definitions with having a great impact on overall expense structures. Clare-Marie Karat approaches cost structuring from the human factor perspective. The major beneficial aspects have certain similarities, such as *Increased sales*, but when viewing from the human effort perspective, beneficial aspects also include *Customer or employee satisfaction and productivity*, found in a case study by Karat (1994, p. 108–109). Other aspects by Mayhew and Tremaine (1994) describe the product lifecycle and the overall cost effects. In the hypothetical project examples, first year and product lifetime *predicted benefits*[5] were compared based on *time value of money models* by Karat (1994, p. 120–139). The results of the comparisons and the examples by Karat et al. show great differences and cost-benefits in the average five-year software product lifecycle where implementations were successful.

In addition to the external benefits, the effects linked to inner organization relations should be underlined, most notably regarding reduced development cycles and various supporting elements. The intangible effects of usability offer prolonged beneficial changes. In development projects there is tendency to promote taking an approach towards increased usability design activity when implemented with careful planning and vision of external or internal benefit. In this thesis, a set of automated

[3] In the context of this study, user-centered laboratory usability tests

[4] In this case, support costs are defined as user support that includes online, phone and training services

[5] In this context, predicted benefits are solely usability triggered cost-benefits

data analysis technique outcomes, aimed directly towards customer HCI behavioral data, could more accurately quantify effects of usability development effort. As opposed to post-release laboratory testing of usability improvements, the approaches introduced in this thesis could provide low cost measurement tools suitable for both usability and software development.

The cost-benefits of the outcome of the thesis can be seen in both tangible and intangible assets. Based on usability study related generalized assumptions mentioned above, achieved benefits from this research outcome could be seen in increased sales, reduced costs and new offerings such as:

- Increased sales due to customer satisfaction and productivity
- Better strategic choices due to statistical help in decision making

Added value and cost reduction could be seen in:

- Reduced development strain due to the efficient allocating of resources
- Reduced development iteration due to finding repetitive behavior models
- Reduced usability prerequisite evaluations for finding valuable issues
- Suggestion based, scalable functionalities based on learning data-sets
- Analysis service offerings, based on sequential data analysis

1.2 Scope of the study and data

This paper focuses on BIM software. In the construction industry, this is a highly growing standard for all types of construction design, detailing, management, and production planning tasks. In this workflow, the stakeholders are working in a 3D environment[6] creating and obtaining information rich data[7] and visualizations. An example of a 3D model environment is shown in Fig. 1. The data created during design processes also cover elements of 4D[8] and 5D[9] which describe time and cost planning of projects (Eastman et al., 2008, p. 24; 359).

[6]Three-dimensional representation of geometric data
[7]Geometric objects that contain attribute data such as *name* and *ID*
[8]Commonly refers to the dimension of the time and schedule constraint in construction modeling
[9]Commonly refers to the dimension of cost in construction modeling

Figure 1: The Halfdan offshore platform constructed in 2007 (above) and Tekla Structures structural BIM model of the platform (below), digital photograph, Saipem S.p.A.

In order to obtain a reasonably sized and less error-prone data set for analysis, the evaluation was focused on structural engineers[10] who were using test application detailing tools for structural detailing. Obtained application event data currently has known missing unmapped entries,[11] but they are not present in the design tools used in this work analysis. The scope of the evaluation can be considered both interesting and useful, as the test application[12] contains over 1000 pre-installed design-tools, the majority of which have no record for usability inspection or early user feedback that could have impacted the tool designs.

Although the evaluation focuses on design tools, the approach can be considered to suit all software related UI. Additionally, the approach may theoretically suit user-created BIM information analysis by opening possibilities to learn from actual user creations, such as structural design and documentation.

The level of generalizability was a significant factor when selecting the methods for analysis. If the methods were suitable for automated pattern discovery with min-

[10] In this context, the designer of a load bearing structure
[11] See missing data entries in Section 4.3
[12] Tekla Structures 2016i Full using Default environment

imal parameter calibrations, they were selected to be used to pinpoint the location of possible usability problems in the available data. Locating problems was further divided into different depths[13] of which were a requirement for extracted event data.

1.3 Research questions and structure

The primary qualitative research questions for this thesis were the following:

1. For what are the found exploratory sequential data analysis (ESDA) usability methods best feasible in contemporary BIM applications?
2. What kind of software usage information is revealed from sequential data?
3. How can the obtained information be utilized in prediction?

The thesis consists of five parts: Theory (Section 2.) will discuss existing theory about both traditional and automated usability testing studies as well as look at pattern detection in general from data analysis perspective. Approach selection and focus (Section 3.) will concentrate on the selected approaches and act as an introduction to the test implementation. Analysis (Section 4.) focuses on the implementation of selected approaches. Results (Section 5.) will discuss analyzed data and explain its impact based on the initial research questions. Conclusions and future work (Section 6.) will review the research questions and answer them concisely. The section will also discuss the further possibilities of the created implementations as well as scalability.

[13] See user task depths in subsection 4.2.

2 Theoretical background

Section 2. discusses the research fields and the discipline in general. The emphasis is on usability studies and the use of automated usability evaluation. Non-automated usability engineering and evaluation is discussed and utilised to adapt automated analysis to existing processes. The section also provides possible approaches towards answering research questions introduced in section 1.3. Towards the end of the section, data analysis terminology is discussed and explained from the perspectives of both usability and data mining.

2.1 Usability studies

ISO standard 20282-2:2013[14] defines usability as *"The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use."* Usability is also commonly defined as a measurable characteristic of used user interface (Mayhew and Tremaine, 1994, p. 1). There is no singular property or value to measure the usability of each interface as the requirements of the interface can vary excessively. In software development, the software is generally intended to be used by a distinct user group that also varies. Nielsen states that usability is traditionally associated with five different attributes, shown in Table 1. (Nielsen, 1993, p. 26–36).

Table 1: Definitions of different usability attributes. Adopted from Nielsen (1993, p. 26)

Learnability	The system should be easy to learn so that the user can rapidly start getting some work done with the system.
Memorability	The system should be easy to remember, so that the casual user is able to return to the system after some period of not having used it, without having to learn everything all over again.
Efficiency	The system should be efficient to use, so that once the user has learned the system, a high level of productivity is possible
Satisfaction	The system should be pleasant to use, so that the users are subjectively satisfied when using it; they like it.
Error Rate	The system should have a low error rate, so that the users make few errors during the use of the system, and so that if they do make errors they can easily recover from them. Further, catastrophic errors must not occur.

As shown in Table 1, Nielsen divides usability into different requirements. Depending on the software and user type, each of the attributes can be valued as the highest. The most typically used attribute is noted to be learnability (Nielsen, 1993, p. 27). In many software systems, users start working with new interfaces and learn

[14]See ISO-20282-2:2013 (2013)

functionalities by trial and error. Additionally, most of the created software does not have the capacity to offer enhanced support services, such as training or other user assistance, making good learnability vital for the software to be accepted by users (Nielsen, 1993, p. 28).

In contrast to Nielsen's attributes, defining usability in measurable characteristics can also be divided into factors describing usability as outcomes. Some notable characteristics are described by Mayhew as *Cognitive, perceptual, and motor capabilities*, *Decreased user training time and cost* and *Decreased overall development and maintenance costs* (Mayhew, 1999, p. 2–4). These characteristics describe different types of benefits. In work by Karat (1994, p. 108–109) and Mayhew (1999, p. 2–4), cognitive, psychological and social aspects are defined as benefits involved with human factors. Decreased errors and training or support costs produce value especially for business product users (Rajanen and Iivari, 2007; Ehrlich and Rohn, 1994). Finally, decreased development, support, maintenance, or other organizational costs can be categorized as directly beneficial usability design for the software organization itself (Mayhew, 1999, p. 2–4).

2.1.1 Usability Engineering

Usability engineering can be described as an engineering field that accounts for usability related tasks during the software development process. Rather than focusing on design, one of the main goals of usability engineering is to analyze and find room for realistic usability improvement regarding assigned goals (Nielsen, 1993; Mayhew, 1999, p. 16–17; 125–131). The work of the usability engineer often deals with finding the best possible solution for the assigned task in order to satisfy both the various usability limitations and the overall software development process (Nielsen, 1993, p. 79–80).

Going through software development projects where interface design aspects are discussed as a single part of the overall development is not considered usability engineering. Usability engineering as a development task must be considered and implemented when necessary throughout the software lifecycle (Nielsen, 1993, p. 71–73). There are multiple lifecycle models that describe entire software lifecycles. One of the first simplified lifecycle models was described by Gould and Lewis as principles in 1985. Their three design principles are *Early Focus on Users and Tasks*, *Empirical Measurement* and *Iterative Design* (Gould and Lewis, 1985, p. 300–301). Early focus on users and tasks encourages the designers to understand user types and their characteristics by differentiating viewpoints that vary between cognitive-behavioral and expected work result. The iterative design principle is considered as an early development phase where users are tested by using empirical methods such as prototypes and simulations. The empirical measurements are then recorded and analyzed for better understanding and development suggestions. The iterative design results in fixing the engineering design using iterative cycles until the solution meets required standards. (Gould and Lewis, 1985, p. 300—303)

In comparison with the simple model by Gould and Lewis, Nielsen (1993, p. 72) has presented more advanced life cycle model shown in Table 2a with similar foundation. The model is more detailed and can be followed as 11 different stages through entire development process. Further detailed definitions can be found in the book *The Usability Engineering Life cycle* by Mayhew (1999). Model by Mayhew (1999, p. 5–9) is also shown in Table 2b, that is dedicated to unifying existing usability engineering techniques and explaining lifecycle models thoroughly.

Table 2: Usability engineering lifecycle model comparison (a) Nielsen, (b) Mayhew. Adopted from Nielsen (1993, p. 72) and Mayhew (1999, p. 5–9).

(a)

1. **Know the user**
   a. Individual user characteristics
   b. The user's current and desired tasks
   c. Functional analysis
   d. The evolution of the user and the job
2. **Competitive analysis**
3. **Setting usability goals**
   a. Financial impact analysis
4. **Parallel design**
5. **Participatory design**
6. **Coordinated design of the total interface**
7. **Apply guidelines and heuristic analysis**
8. **Prototyping**
9. **Empirical testing**
10. **Iterative design**
    a. Capture design rationale
11. **Collect feedback from field use**

(b)

1. **Requirements Analysis**
   a. User Profile
      i. Contextual task analysis
      ii. Usability goal settings
      iii. Platform capabilities and constraints
      iv. General design principles
2. **Design/testing/development**
   a. Work reengineering
   b. Conceptual model design
   c. Conceptual model mock-ups
   e. Iterative conceptual model evaluation
   f. Screen design standards
   g. Screen design standards evaluation
   h. Style guide development
   i. Detailed user interface design
   j. Iterative detailed user interface design evaluation
3. **Installation**
   a. User Feedback

The focus of the thesis is on finding the problematic areas of a released product by recognising repetitive behavior. The basis is in the overall usability engineering process *Iterative design* found both from Nielsen's and Mayhew's model examples. Mayhew describes iterative design as *"To further refine the product Detailed User Interface Design"* (Mayhew and Tremaine, 1994, p. 339). Although the original purpose of iterative design is to apply it to the empirical testing of unreleased products, the same method applies to existing development. As potential methods, Nielsen suggests utilising user event log streams for problem discovery, showing where users might have encountered usability related problems, such as time waste and use disruption (Nielsen, 1993, p. 105).

Using data collection during the iterative design process is described to be one of the key factors of obtaining information about time waste (Nielsen, 1993; Mayhew and Tremaine, 1994, p. 106–107; 340). Time waste is also of interest from the customer's perspective, as it's easily measurable and can be transformed into costs. Using event stream to pinpoint time waste could be implemented for the large

audience without user disturbance and with minimal costs due to the automated nature of the process. Finding priorities from possible automated discovery outside of their statistical occurrence, such as fixing time and expense, is left to the usability team. The existing design tools used in test application are primarily indented to be technically functioning correctly. This should allow the design teams to suffice without major interface changes to impact usability. Other measures, such as user satisfaction *surveys* can also be used to some extent, but is not prioritized, as test subject attendance is too low for validation when using surveys.

2.1.2 Usability evaluation

Usability evaluation is the general definition of product evaluation or testing using different techniques that involve either the analysis of user and product interaction or the direct analysis performed by experts (Dix et al., 1998; Nielsen, 1993, p. 319; 155–163). The goal of usability evaluation is to locate usability problems in different user interfaces. One of the fundamental, yet challenging bases of usability evaluation is reliability and validation. As the subjects are typically individual users, test subject results can greatly vary from each other (Nielsen, 1993, p. 43–48, 165–167, 177–180).

Evaluation is commonly divided into two distinct types. Usability testing consists of all testing that contains user involvement. Analyses performed as expert evaluations or walkthroughs are not considered as usability testing, but are identified as usability evaluation in general (Rubin, 1994, p. 21–22). In some types of evaluation, subject specialised experts from fields such as usability or software systems can evaluate or describe usability without any user involvement (Nielsen, 1993, p. 155–157).

The expert evaluation method that is based on a heuristic systematic approach is a common evaluation method. The evaluator, who is a usability specialist or user, performs the evaluation of the assigned product. Analysis is performed according to heuristic rules and possibly the intended user group domain perspective (Rosenbaum, 1989, p. 210). Instead of the review process that follows a certain checklist or preassigned goals during expert reviews, the evaluator is required to understand heuristic principles; however, they are not forced to follow specific rulings. Using expert evaluation is seen as cost-effective in various cost-benefit analyses (Muller et al., 1993; Nielsen, 1993, p. 185; 160).

Systems that require expert domain knowledge to be operated tend to create more false alarms and overall noise when evaluated. This can have a direct effect on the usefulness of the method due to validity and reliability. The evaluators decide if a usability issue has been found, using their own understanding (Nielsen, 1993, p. 155–157). As a result, the found issues vary depending on the expert's level of knowledge. Using multiple experts and obtaining evaluators with high knowledge reduces uncertainty in expert reviews.

Most usability evaluation methods can be categorized under *Ethnographic Research*, *Usability Testing*, *Expert or Heuristic Evaluations*, *Walkthroughs* and *Surveys* (Rubin, 1994, p. 16–20). Additionally, Nielsen describes a more detailed *Thinking Aloud*[15] method as "probably the single most valuable usability engineering technique" (Nielsen, 1993, p. 195). In a more recent paper by Ivory and Hearst, usability evaluation is divided into five different classes (Ivory and Hearst, 2001, p. 473). The classes are *Testing*, *Inspection*, *Inquiry*, *Analytical modeling* and *Simulation*. The paper focuses on automation techniques and is considered to be a useful addition to modern usability evaluation (Ivory and Hearst, 2001, p. 472). Other techniques that fall under these categories as traditional methods or automated techniques are further explained in this section.

Ethnographic Research gathers information about user activity in an environment that is familiar to the user. In software development, the test user should use a known software product in familiar surroundings (Rubin, 1994, p. 16). Using these surroundings during any tests with direct user involvement can help to minimize unwanted disruption from unfamiliar circumstances. The research method is qualitative, and is used to obtain user information during product interaction (Rubin, 1994, p. 16). In this context, the methods are used to create a more natural test environment.

Usability Testing involves direct user interaction techniques and methods. The idea of having real users testing the product is described as a fundamental part of usability testing (Nielsen, 1993, p. 165). Basic usability testing scenario yields a participant and an evaluator. During testing, the user performs tasks using the system or product and tries to achieve assigned goals or tasks (Ivory and Hearst, 2001, p. 477).

Walkthroughs are considered as expert evaluation, used to explain user necessity or the individual user's way of working, and to eliminate the problematics found by using iterative cycles (Rubin, 1994; Nielsen, 1993, p. 19; 155). Walkthroughs are a part of the inspection classification described by Ivory and Hearst, but also refer to techniques that contain preassigned criteria and use of various heuristics (Ivory and Hearst, 2001, p. 473, 475–476).

By using a walkthrough, the designer is able to visualize the familiarized user group ability to succeed in an assigned task that involves system interaction (Rubin, 1994, p. 19). Usability walkthroughs are commonly used in two different methods: A *Cognitive Walkthrough* (CW) describes user interface by asking how the tested user interface functions with new users. Test steps involve at least a *Set of goals* that the test subject is trying to achieve, *A search and selection of correct actions* to proceed during use and *Performing the selected actions* in order to complete the set of assigned goals. An organizer will walkthrough the product, and the evaluator will ask questions from the user perspective (Rieman et al., 1995, p. 387–388). If the users form alternative paths or breakdowns, they are evaluated, and changes

[15] See Nielsen (1993, p. 195–200)

are made into the system where necessary. The process is then evaluated again to review improved performance and the necessity of the additional improvements. A *Heuristic walkthrough* (HW) has many similarities to CW, but rather than trying to find problems or breakdowns, the usability is evaluated by a set of heuristics. The most well-known form of heuristic principles is by Nielsen (1993, p. 115–163). There are various other sets of heuristics created. Introduction to these sets can be found as a collection in Nielsen and Molich (1990, p. 249–250).

Surveys or Inquiries as usability testing methods simply ask the users questions related to product usability and giving feedback (Nielsen, 1993; Ivory and Hearst, 2001, p. 209; 473). Survey methods, such as questionnaires, are indirect and measure individual opinions (Nielsen, 1993; Rubin, 1994, p. 209; 18). A common survey method is the System Usability Scale (SUS) created by John Brooke in 1986. SUS is a standardized question set regarding system usability and has been validated by various studies (Brooke, 2013, p. 38).

2.1.3 Evaluator importance

Although automated usability analysis approaches are able to reveal usability related information without direct evaluator presence, the evaluator is still often responsible of choosing what to evaluate and how to interpret results. By utilizing the implementations based on this thesis, the evaluator is responsible for filtering and filtering model functionalities. As the detection of user behavior and possible model examples in section 3.1 and the implementation in Section 5.1, justification of evaluator importance is introduced. There is only a single model creation for the testing purposes in this work. Based on the following section, it should be decided by the reader as what kind of evaluator knowledge is required to select, utilize and interpret revealed information.

Various models that have been created for the purpose of finding usability problems are based on different methods and human factor scaling. Measuring the performance of different approaches can be challenging and does not always provide stable, measurable information (Muller et al., 1993, p. 185–186). Different methods tend to work for certain products and user groups better than others. For instance, expert software systems and BIM software require expert knowledge in order to understand and find important usability problems.

In terms of finding usability problems, the difference between user and evaluator outcome was introduced in Rasmussen et al. (1989, p. 518–520), who described that individuals are affected by three factors that they possess. These factors are *Skill-based*, defined as the ability to recognize attention and the right signals in the interface, *Rule-based*, defined as the ability to respond to ongoing procedures, and *Knowledge-based*, measuring the ability to create mental models of the product. Nielsen et al. conducted empirical comparison studies with similar findings, suggesting that the amount of usability problems found during expert evaluation may vary significantly depending on the level of knowledge of the evaluator (Nielsen,

1992, p. 375). Furthermore, Nielsen et al. created a mathematical model based on heuristically evaluated test cases. The model describes the amount of evaluators required to find a certain percentage of usability problems (Nielsen and Landauer, 1993, p. 206–213).

System evaluation is efficient with a relatively small number of evaluators or users. On average, using five evaluators results in 75% of usability problems being found. This is shown in Fig. 2., that is based on the evaluation model and values[16] from heuristic evaluation cases in Nielsen and Landauer (1993, p. 206–213). There is a close reference to user testing and a similarity with the heuristic approach when examining mixed user and evaluator group median values. The performances of user testing and heuristic evaluation are the same when using six individuals. Since the initial models defined by Nielsen et al., there have been newer ones, such as Schmettow's model that discusses the effects of *heterogeneity* and *sample size* affecting the original model (Schmettow, 2008, 2012). When looking at the model, the effects of complex expert software evaluation might cause a decrease in mean values as diversity increases software heterogeneity. By using the models provided by Schmettow, random variation could be calculated to provide a more accurate problem discovery percentage (Schmettow, 2008, p. 89).

Nielsen et al. grouped the performance of evaluators into three different categories that consist of *Novice evaluators, Regular specialists and Double specialists*. The group types were described as 1) novice as not familiar with usability but otherwise normal user, 2) regular specialists as usability specialists that have experience from usability design and evaluation perspective but no experience in the actual product domain, and 3) Double specialists that have experience in both usability and product domain and are familiar with the product (Nielsen, 1992, p. 376). The statistics in Fig. 2. show significant difference between evaluation performance between different knowledge levels. The requirements to achieve generalized percentage yielding 75% of found problems are included for reference. The novice group reaches 71% with 14 evaluators. The regular group requires three to five evaluators for 74–87% and the double specialist requires two to three to reach 81–90%. The percentages are recommended by Nielsen for successful evaluation (Nielsen, 1992, p. 376–377).

Although the heuristics are not used in this study, the findings of Nielsen et al. and Rasmussen could be considered important due to their categorization of users and evaluators into groups giving some metric directive for problem discovery. As described, the mean values obtained from heuristic and user testing provide results close to each other. However, when evaluating complex expert software, the likelihood of finding problems can be notably lower. Somewhat similar findings have been found when viewing creation-oriented software Trimble Sketchup (Akers et al., 2012, p. 2). Akers found that the number of test participants required in order to find 75% of usability problems was estimated as 30 users. As with

[16]See p. 207 "Heuristic Evaluation" in Table 1. (Nielsen and Landauer, 1993, p. 207)

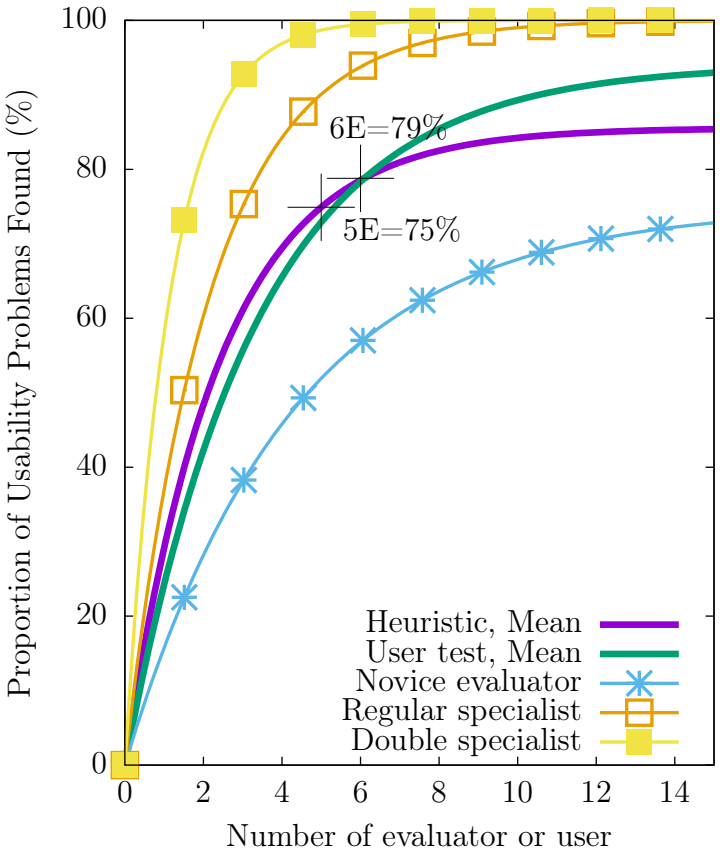


Figure 2: Difference between novice evaluator, regular and double specialist usability problem discovery when performing heuristic evaluation. Based on the mathematical models from Nielsen and Landauer (1993) and Nielsen (1992).

expert software, the diversity of applications and the large number of possibilities from approaches to goals contributed to difficulties in finding problems in creation-oriented software (Akers et al., 2012, p. 3). As BIM software requires expert field knowledge for a wide spectrum of structural design domain, in order to be able to recognize user disruption from user behavioral sequences, evaluator grouping should be taken into consideration. This could be similar to Schmettow's heterogeneity related corrections to the model created by Nielsen and Landauer (Schmettow, 2008, p. 97). The use of double specialists in expert software evaluation increases the amount of overall usability problems found.

In this thesis, all of the event data was obtained from users as an automated background process. User are generally non-conscious about the ongoing data capture and are not obligated to any direct involvement. This should minimize any altering of behavioral effects caused by unfamiliar circumstances. The data could be seen as neutral raw direct user activities.

Users typically perform detailing work using their own systems and personalized. Among all of the data, there is a possibility that some of the work has been done as testing or in unfamiliar circumstances. The effects of this kind of data could alter the resulting data with a smaller sequence sample size. In larger sequence samples, the results of the analysis reveal the most common occurrences. The users who do testing for their own organization are likely to play a minor role in the ordinary workflow. Thus this kind of data should not be dominant, and as a result, it ought not be seen in the highest occurring patterns.

2.2 Data analysis

Data analysis is a process that gathers and analyzes information from any target activities to some form of digital entity. Target activities can be anything that includes a need for data driven decision making (Myatt and Johnson, 2014, p. 1–2). The field of data collection processing has grown alongside the information technology evolution. The first collection of data was shown around the 1960s as primitive file processing (Han, 2012, p. 3–4). By the 1980s, interest in data analysis began to grow highly due to the advances in *database management systems*, and the creation of *data warehouse*[17], and different *Online Analytics Processing* tools - all powerful for data handling and modeling (Han, 2012, p. 3–4). As the analysis process contributes to many fields, it can be described as multidisciplinary (Cuesta, 2013, p. 7–8). Hector and Myatt et al. describe the basic structure of the data analysis process very similarly as shown in Table 3. In Han (2012, p. 7), the model has similarities but emphasizes a data mining perspective with an elevated focus in knowledge discovery. In this process, preparing the data is shown as model cleaning, selecting and transformation.

Table 3: Data analysis process model. Adopted from Myatt and Johnson (2014, p. 4). Different to Myatts model, Cuesta (2013, p. 11) also has separate distinctions of data exploration, predictive modeling and visualization of results.

1. Problem definition
2. Data preparation
3. Analysis
4. Deployment

Problem definition covers areas that are yet to be solved. As analysis implementations may require careful settings for a shallow scope, the problems need to be clearly structured and planned for constraining and keeping work focused (Myatt and Johnson, 2014, p. 5). This does not mean that the resulting information is always known beforehand, but rather assumed to answer broader questions. In popular work by Hartwig (1979, p. 3), regarding exploratory data analysis, this is

[17] In this context, a system for data analysis and reporting. See about data warehouse in Han (2012, p. 125–128)

described as the discovery of new relationships between known assets. In this thesis, the research question structures closely followed a more open ended exploratory analysis approach.

Data preparation focuses on creating definitions of the data to be collected and structured. This includes characterizations, cleaning, transforming, and portioning tasks for further processing (Myatt and Johnson, 2014, p. 4). This step is often the most time-consuming step in successful analysing. Data should be in a format that can provide answers to the given research questions most efficiently (Myatt and Johnson, 2014, p. 8). Typical drawbacks during data preparation include issues such as invalid, ambiguous, out-of-range, or missing values (Cuesta, 2013, p. 12). The characteristics of a good data-set are listed as being complete, coherent, unambiguous, countable, correct, standardized, and non-redundant (Cuesta, 2013, p. 12). Existing AUF-data[18] is in raw format, but is cleaned and checked to follow the above mentioned characteristics. Missing and invalid data is reported and explained in section 4.3.

Analysis goes into finding and implementing the correct analysis methods for structured data. Myatt and Johnson (2014, p. 9–10) explain analysis in three distinct categories that involve *summarizations and statements*, identification of important content such as *facts, relationships, anomalies or trends* and *mathematical model creation* to encode data relationships. In this thesis, the focus is on the last two categories. The definition of the analysis type follows the qualitative domain, defined as *"numerical measurements expressed in terms of numbers"* (Cuesta, 2013, p. 13).

The classification of data and the corresponding approach for the analysis regarding outcomes is well established among known categories. In Cuesta (2013, p. 13), the listing of common classifications among data analysis refers to *Categorical* as classification, *Numerical outcome* as regression and *Descriptive* as clustering model outcomes.[19] In this study, the classification method scope follows the use of *Frequent patterns*, where attributes in existing data can be revealed to occur in the same order in multiple instances (Han, 2012, p. 415–416). One of the first implementations among the classification was described in Agrawal and Srikant (1994), related to *"Basket data"*, that is currently typically recognized as *Market basket analysis* (Han, 2012, p. 244). This also lead to the creation of the well known *Apriori*[20] algorithm for mining frequent itemsets. The ESDA based thesis implementation mostly utilizes similar, sequential pattern mining approaches.

Deployment is the last phase mentioned by Myatt. In this phase, the resulting data is carefully translated to give the client beneficial value in forms of reports or direct business impact measurement (Myatt and Johnson, 2014, p. 11–13). Among

[18] See Fig. 6. for an example of raw data

[19] See classifications also in Bramer (2007); Han (2012); Scarpa and Azzalini (2012); Myatt and Johnson (2014) for common and other more advanced classifiers.

[20] See Apriori in Agrawal and Srikant (1994)

the different available methods on quantitative or qualitative summaries, the deployment result can vary from numerical visualization to descriptive results. Cuesta (2013, p. 15) emphasizes the importance of visualization as it usually represents the results for all further data related decision making processes.

2.2.1 Exploratory data analysis

Exploratory Data Analysis (EDA) is an approach that implements various data-analysis techniques without a strict hypothesis or assumption of what is going to be revealed. In EDA, one of the main goals is in investigating data for a characterization that would ultimately allow the extraction of patterns (Behrens, 1997, p. 131–132). Major contribution to definitions of the EDA approach was in the book *"Exploratory Data Analysis"* by John W. Tukey (1977).

From a more usability centered perspective, EDA is seen as broad requirements of how data is handled with the selection of different techniques (Fisher and Sanderson, 1993, p. 34–35). In Sanderson and Fisher (1994, p. 264–265), EDA is described as an approach that reveals quantitative observational data for better qualitative understanding. In Hartwig (1979) and Tukey (1977) as cited in Sanderson and Fisher (1994), EDA is based on three principles:

Continual openness and re-expression. Data is explored for possible patterns with simple statistics before categorization and the use of a certain model. Finding patterns might require several iterations to "smooth" data. In contrast, the rest of the available data is considered, yet unexplained. Transformations applied to the data are considered essential to discover new patterns.

Initial skepticism. As the data does not always contain a unique and correct numerical summary due to smoothing, the data is not believed to be statistically fully correct. It is also assumed that EDA preferred statistics are more sensitive to the bulk of data rather than outlying points.

Exploration versus confirmation. The EDA approach supports data exploration to generate hypotheses. These can later be analyzed and fully validated with different statistical methods.

The central implementations of EDA are related to graphical methods, revealing data correlation in visualized format. Non-graphical methods deal with calculations of data, using different statistical methods (Myatt and Johnson, 2014; Behrens, 1997, p. 10–11; p. 135). The following section *"Sequential data analysis"* and used pattern detection approaches such as *Predicting* and *Sequential pattern*[21] or *Frequent itemset*[22] -mining can be categorized under EDA.

[21] See Section Sequential pattern mining 2.2.3

[22] See in SPMF (Fournier-Viger et al., 2014b, Frequent Itemset Mining)

2.2.2 Sequential data analysis

Sequential data analysis is a wider term for activities related to recorded data elements, containing ordered sequential information. This information can be analyzed through a variety of methods. Popular applications typically focus on discovering useful patterns in databases (Sammut and Webb, 2010, p. 902). Approaches under *Sequential pattern* or *Frequent itemset* mining that emerged during the 1990s have gained popularity due to widespread applications, such as *Market Basket Analysis*, *DNA sequencing in biomedical applications* and *analysis of user actions* (Fournier-Viger et al., 2017; Sammut and Webb, 2010, p. 55, p. 902).

In this thesis work, Sequential Data analysis is strictly considered as the result of the data that contains sequential information. As an example of such data, Fig. 3 shows the process model of simple decision making. From point A or B all decisions lead to point C. Depending on the decision in point C, there will be a return to either point A or B. Recording points A or B could example result in a observable sequence such as:

$$AAABBAAB.$$

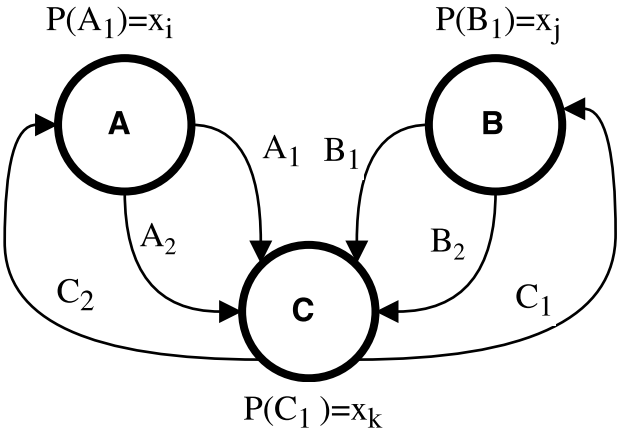


Figure 3: Example of a small decision making process that formulates sequential data. Depending on nature of the decision, there should be variance in forming probabilities $P(A, B, C) = x_{i,j\ ,k}$.

From the Usability Sciences' perspective Exploratory Sequential Data Analysis is a working term for a loose set of data analysis activities in human sciences that deal with recorded data (Fisher and Sanderson, 1993, p. 34). Related techniques provide solutions to *detecting sequences*. During ESDA activities, data such as audio, video, or any other data source is directly analyzed. Sequential Data Analysis also refers to the use of techniques where all data relations and dependencies are preserved (Fisher and Sanderson, 1993, p. 34–35). The techniques considered as a part of ESDA are not defined clearly by Fisher and Sanderson. Some distinctions can be found in

the work of Hilbert and Redmiles (2000), Cuomo (1994) and Olson et al. (1994). Based on the distinctions, the applications of ESDA with the highest interest for this particular thesis and the subjects for further study are evaluated in section 2.4.

2.2.3 Sequential pattern mining

The first definition created for sequential pattern mining was by Agrawal and Srikant (1995, p. 5): *"Given a database of sequences, where each sequence consists of a list of transactions ordered by transaction time and each transaction is a set of items, sequential pattern mining is to discover all sequential patterns with a user-specified minimum support, where the support of a pattern is the number of data sequences that contain the pattern."*

There are two common subtypes of data: *Time-series* and *Sequential*. Time-series is a time ordered list of numbers. A sequence is an ordered list of nominal values, as shown in Fig. 3. The main interest of sequential pattern mining is in finding all the *subsequences* in a set of sequences. A subsequence is called a *frequent sequential pattern* if it appears more than once in a set of sequences (Fournier-Viger et al., 2017, p. 55–56). For test application data, this thesis concentrates on exploring sequential pattern approaches as the target data is a collection of user events. Additionally, the use of *frequent itemsets* approaches is excluded due to data containg multiple identical items in a itemset that is generally not accepted.

The problem of sequential pattern mining is expressed in Mooney and Roddick (2013); Fournier-Viger et al. (2017, p. 19:3, p. 56–57) defined as: Let there be a set of items, such as nominal values $I = i_1, i_2 ... i_m$. An *itemset* is an nonempty unordered collection of items, such as $X \subseteq I$. Itemset X is considered to be length of k or a k-itemset when it contains a k amount of items. Without loss of generality, it is assumed that the items in the itemset are in lexicographical order. An ordered list of itemsets $s = \langle I_1, I_2, ... I_n \rangle$ is called a *sequence*, where $I_k \subseteq (1 \leq k \leq n)$. As an example, consider the following sequences s_1 and s_2:

Table 4: A set of sequences s_1 and s_2

$$s_1 = \langle \{a, b\}, \{a\}, \{b\}, \{a, b, c\} \rangle$$
$$s_2 = \langle \{a, b\}, \{a, b, c\} \rangle$$

In sequence s_1 and s_2, the letters represent a person's decision making process. The sequence s_1 contains four different decision itemsets, some of which contain multiple simultaneous choices, such as a, b and a, b, c. The length k in a sequence is the total number of individual items of a, b or c, for instance $k = |I_1| + |I_2| + ... + |I_n|$, and thus $k_{s1} = 7$.

When calculating all the possible subsequences that support both s_1 and s_2, the result is a total of 28 sequential patterns, five of which are shown below in Table 5.

The *support* of a sequential pattern s_n is the number of sequences where a pattern occurs divided by the total number of sequences in a given sequence set, or *Sequence Database* (SDB), thus $sup(s_n) = |\{s \mid s \sqsupseteq s_b \wedge s \in SDB\}|$ (Fournier-Viger et al., 2017, p. 57). When looking at the subsequences below, all $sup(s) = 2$. The longest sequential pattern found is $\langle\{a, b\}\{a, b, c\}\rangle$ found both in s_1 and s_2.

Table 5: five sequential patterns found from set of sequences s_1 and s_2

$\langle\{a, b\}\rangle$
$\langle\{a, b\}\{a\}\rangle$
$\langle\{a, b\}\{a, b\}\rangle$
$\langle\{a, b\}\{a, c\}\rangle$
$\langle\{a, b\}\{a, b, c\}\rangle$

This thesis later utilizes mining variations to *maximal*, *closed* and *top-k* sequential patterns.[23] The variations are discussed in more detail in Section 3.3. The starting points for each variation the remain same as defined in this section. I have specifically selected to use the SPMF open source data mining library (Fournier-Viger et al., 2014b) as the main portion of its available mining methods are close to the proposed ESDA or SDA related approaches inspected in Section 2.4.

2.2.4 Sequence predicting

The modeling and prediction of paths can be considered to be in the same continuum as sequential pattern mining. Sequence predicting is considered to be a part of the machine-learning field, in the form of prediction or decision tree based learning (Alpaydin, 2014, p. xvii, p. 214–215). The majority of applications have emerged from disciplines such as *DNA sequencing, financial engineering, robotics* and *speech recognition* (Sun and Giles, 2001; Alpaydin, 2014, p. 1–2, p. 3–13). Currently, learning from data, for instance *machine-learning*, is a popular area of data-analysis, particularly in online business applications (Alpaydin, 2014, p. xviii).

The problem in sequence prediction related to this work is defined in Sun and Giles (2001); Gueniche et al. (2013, p. 2–4, p. 2) and can be expressed as follows: Consider again the two sequences in Table 4., or more. In a $SDB = s_i, s_{i+1}..., s_n$ and the next items $s_{n+1}, ...s_{n+1+k}$ where $(1 \leq i \leq n)$ and $k > 1$. Given $s_i, s_{i+1}..., s_n$, we want to predict either s_{n+1} or $s_{n+1}, ..., s_{n+k}$ by training sequences and building a prediction model.

[23] See closed and maximal sequence detection examples in Sections 3.3.1 and 3.2.3. See non closed, top-k in 3.3.2, or from all in more depth definitions in Fournier-Viger et al. (2017)

In this study, a *Compact Prediction Tree* (CPT) based prediction model is used together with the resulting sequential pattern data. The approach is described more in depth in Section 3.3.3. Various different models exist, but CPT is considered directly implementable for test application purposes and is considered superior to similar[24] *Markovian*, *All-k-Order Markov Chains* (AKOM)[25] or *First order Markov Chains* (PPM)[26]

2.3 User events and data

Many of the techniques proposed by Hilbert and Redmiles (2000) focus on finding user problem areas or reducing the domain area to pinpoint user problems in existing software systems. However, this requires the software or the user to be monitored in some way to produce repetition for automated base data collection. For the Tekla Structures software that is primarily analyzed in this thesis, a previously created feature of collecting user feedback event data in the software's User Feedback Program (UFP)[27] is used as one of the primary elements. In Siochi and Ehrich (1991), repetitive events in user sessions were analyzed to find possible usability problems. This included the use of the ESDA technique that recognizes sequences directly from backtracked event data transcript, which contain a time-stamp (Siochi and Ehrich, 1991, p. 4). The study introduced the use of Maximal Repeating Patterns (MRPs) in usability analysis. MRPs log the users' natural sequences of tasks. The research hypothesized that registered, repeated sequences of actions indicate a task rather than random sequence (Siochi and Ehrich, 1991, p. 4–5). Focusing on finding performed tasks as repetitively occurring sequences is a possible method of finding usability related information.

Similar to the functionality of MRPs, techniques called Fisher's Cycles (FC) by Carolanne Fisher and Lag Sequential Analysis (LSA) first introduced by Gene Sackett also extract information from event data (Hilbert and Redmiles, 2000, p. 404). The main difference between MRPs compared with Fisher's and LSA is that the investigator can assign a particular event of interest. In Fisher's Cycles the investigator can specify a beginning and ending event to identify the content as a sequence. In LSA, a 'key' and 'target' event is assigned, reporting the distance to reach the target (Hilbert and Redmiles, 2000, p. 404–405). The techniques introduced above are explained more in depth in section 3.2.

[24] In this context, precise sequence prediction models

[25] See Pitkow and Pirolli (1999) about AKOM and for performance comparison against test datasets Gueniche et al. (2015, p. 9–11)

[26] See Alpaydin (2014, 2.1 Learning Markov Chains) and for performance comparison against test datasets Gueniche et al. (2015, p. 9–11).

[27] UFP-program is voluntary user contribution program where unreleased material can be tested by user before final release

The use of ESDA techniques creates various challenges discussed by both Hilbert and Redmiles (2000, p. 407) and Cuomo (1994, p. 9). The challenges of these techniques mainly deal with high noise and a large number of occurring natural events. The techniques do not reveal the exact usability problem, but rather where the problems might occur and where to look for them. Also, defining the correct parameters that indicate usability problems for techniques such as Fisher's Cycles or LSA can be very difficult and time consuming. For example, Fisher's Cycles typically requires a set of parameters on what to look for in each specific case. The proposed applications in automated analysis should work properly when used together with more traditional approaches. Automated analysis use is highlighted especially while in search of usability problems.

An interesting use of pattern related techniques together with the more traditional methods, such as *Thinking-aloud* protocol and *Video capture* was used in Akers et al. (2012, p. 16). Akers et al. introduced a technique that implemented *Retrospective Thinking-Aloud* (RTA) for participant video capture and the use of backtracked event data fixed to an equivalent time-stamp (Akers et al., 2012, p. 38–44). In the think-aloud method, the researcher will ask the participants to perform tasks while verbalizing their thoughts in a controlled environment without distractions (Nielsen, 1993, p. 195). This is also referred to as a *concurrent* method. In the retrospective variant, the participant will comment the process in a similar way but shortly after the actual test of using the software so that the trace of the participants' workflow remains in short term memory (Ericsson and Simon, 1993, p. 16). Akers decided to use the retrospective method after finding that the participants have difficulties with verbalizing their thoughts (Akers et al., 2012, p. 40). Issues related to the loss of important information in RTA is discussed on a theoretical level in Ericsson and Simon (1993), although newer studies such as Guan et al. (2006) indicate that usability researchers can trust the information provided by the participants. In the research by Akers et al. (2012), single target events were recorded and video episodes were created from these event instances for an RTA review.

2.4 Automated evaluation in usability studies

The use of computer automation in usability has repeatedly been a field of interest due to its potential to yield measurable results, its high visibility of problems, and the available cost-benefit[28]. Usability evaluation can be considered expensive when comparing time consumption and human efforts (Ivory and Hearst, 2001, p. 470). In this thesis, implementing evaluation approaches that perform computer automated routines to discover usability related behavior offers an advantage by introducing a set of measurable distinctions and trying to drastically reduce the need to locate problems in software. Methods introduced in this section are primarily intended for existing complex systems that contain different creation oriented tools often required in the BIM domain.

[28] See section 1.1 on cost justification.

A summary of the evaluated techniques for the purposes of this thesis work can be seen in Table 6. The latest approaches and collections found suitable for this thesis, more specifically event data analysis of software, were from Akers et al. (2012), Ivory and Hearst (2001), Hilbert and Redmiles (2000), Cook and Wolf (1995) and Cuomo (1994). There are various newer studies on HCI *'Automated usability analysis'* that deal with pre-defining an optimal path or that are exclusive for either Web or Mobile applications, only to mention a few examples. Due to this, suitable recent studies seem to be few and far between.[29] However, the principles of sequential analysis of usability data have remained somewhat static. The techniques introduced are quite universal and can be implemented to various different purposes without without major changes or the risk of unrecognizability.

The different usability problem detection techniques were analyzed using available studies. A total collection of 23 approaches were inspected. Other techniques considered as irrelevant were directly left out of thorough inspection due to a very limited minimal scope requirement. Additionally, two languages for event detection were inspected but not implemented. The languages were General Event Monitoring Languange (GEM)[30] and Task Sequencing Languange (TSL).[31]

In Table 6., *Pattern detection* denotes if the technique can successfully detect sequences from undefined types of data without a given optimal source or any other comparative target sequence. As an exception, single start and ending fixed event trigger definitions are allowed. *Event data compatibility* denotes if the technique can handle event based log data. Specified event data used in this thesis can be seen in Fig. 6. and it is further discussed in section 4.3. *Generalizable for a broad range of tools* denotes BIM detailing tools in software analyzed in this thesis test application. The technique must be able to capture repetitive patterns from a wide range of used detailing tools with minimal presets. The presets are divided into two allowances that follow parametrized beginning or ending triggers for sequence detection and matching behavioral *problem model types*[32] for summary categorization.

Different kinds of automated applications especially for *Windows, Icons, Menus and Pointers* (WIMP) have been used in numerous usability analyses (Ivory and Hearst, 2001, p. 502). Task model[33] based *Expectation-Driven Event Monitoring* (EDEM) as a full monitoring system enabled usability monitoring by using predefined sequence path called "Expectation Agents" (Hilbert and Redmiles, 1998, p. 2–3). By recording the user workflow path using log file analysis, EDEM will notify designers if a sequence path was broken by a user (Hilbert and Redmiles,

[29] Based on searches to databases such as Association for Computing Machinery (ACM) digital library (dl.acm.org), Elsevier ScienceDirect (sciencedirect.com), IEEE Software (ieeexplore.ieee.org) and Google Scholar (scholar.google.com)

[30] See Mansouri-Samani and Sloman (1997)

[31] See Rosenblum (1991)

[32] Problem models refer to predefined event patterns that are identified as usability problems. See section 3.1 and Table 7. about problem models and type examples

[33] A task model analysis is usually performed by the software or system designers and try to investigate users activities to reach certain goal. Lecerof and Paterno (1998, p. 866–867)

Table 6: Summary of computer aided techniques for usability analysis. Consists of study collections of Ivory and Hearst (2001), Hilbert and Redmiles (2000), Cook and Wolf (1995) and Cuomo (1994). Empty fields denote an unknown status. Probable label denotes an elevated level of acceptance but with uncertainty. Based on the inspections, Fisher's Cycles, LSA and MRP are the most suitable for given conditions.

Method type	**Pattern detection**	**Event data compatibility**	**Generalizable for a broad range of tools**
ADAM	No	Yes	No
AMME	Yes	Probable	No
DRUM	No	Probable	Yes
EBBA	No	Yes	Probable
EDEM	No	Yes	No
EMA	Probable	Yes	No
Fisher's Cycles	Yes	Yes	Yes
FSM	No	Probable	
IBOT		Probable	No
I-Observe	No	Probable	Probable
KALDI	Yes	No	Yes
LSA	Yes	Yes	Yes
MIKE	No	Yes	Yes
MRP	Yes	Yes	Yes
QUIP	Yes	No	Yes
RemUSINE	Probable	Yes	No
RNet	Probable	Probable	No
TOP/G	Yes	Yes	No
UsAGE		Probable	
USINE	Probable	Yes	No
VISVIP	No	Yes	Yes
YEAST	No	Probable	

1998, p. 4). Diagnostic Recorder for Usability Measurement (DRUM) is another automated method using metrics[34] based evaluation method that uses synchronized video recordings, observations and event data created by Macleod and Rengger (1993). DRUM allows the logging of single events or larger sequences in which the designer is interested. Based on the logged recordings and statistics collected, the usability evaluators are able to see specific video entries of these event sequences without going through full recordings (Macleod and Rengger, 1993, p. 1–8).

[34]In this context, metrics is defined as quantitative measure of event based data

The task model or the methods that follow *Comparing Sequences* approaches provide close similarities in workflow to what is proposed in this study. USINE is a tool present in the work of Lecerof and Paterno (1998). USINE functions by creating task model based statistical results from user event logs. The method provides statistics when the user differs from the task model path. The task model path is created by using a special tree editor. USINE tool records assigned event data from a tree task model, including mouse clicks, widget names and time relations (Lecerof and Paterno, 1998, p. 869–887). The basis of the method was introduced by Lecerof and Paterno (1998, p. 869), and it is somewhat visible in the evaluation models of this thesis, seen especially in Fig. 5. Using the comparison of expert created task models such as in USINE is not implemented in this study as it is not possible to determine specific user goals fully in advance.

Other metrics such as *Automatic Mental Model Evaluator* (AMME)[35] or task model approaches such as iBOT[36], *Quantitative user interface profiling* (QUIP)[37] or KALDI[38] are not relevant for the generalized analysis in this work. Finding optimal paths to perform user task with Petri Nets[39] or comparing user performances by using automated log analysis does not apply to analyzing a varying amount of detailing tool UIs. Nevertheless, these approaches could be suitable for further single tool usability analysis, and do not exclude the use of learning based methods.

[35] See Rauterberg (1993)
[36] See Zettlemoyer et al. (1999)
[37] See Helfrich and Landay (1999)
[38] See Ivory and Hearst (2001, p. 481–482)
[39] Mathematical model that try to imitate user actions, see Rauterberg (1993)

3 Selected approaches for implementation

In this section, the selection and approach for implementation described in section 2., are explained in more detail. The beginning of the section deals with software use behavior model candidates for usability disruption categorization. The latter part of the section goes into the automation of usability and introduces a model for automated event data capture. At the end of the section, the selected usability study automation techniques and their corresponding mining approaches are explained.

3.1 Pattern recognition using behavior models

This thesis introduces an approach of evaluating the feasibility of event data and software use behavior extraction in BIM detailing workflow. To my knowledge, no similar research appears to be done where usability study driven workflow sequence detection and sequence pattern recognition is created and tested against pre-existing large data-set in BIM software domain. Creating a basis for raw pattern extraction using techniques that emphasize behavioral repetitive patterns is considered to be of interest for this study. This section introduces models to evaluate and categorize such data extractions, but additionally considers that the selected automated techniques must not be limited to found recognition models only. The selected techniques must also be adaptable in order to be used by evaluators according to their own filtering rules and interest.

As introduced briefly in sections 1.1 and in 2.4, there are various types of methods and techniques regarding computer automation and finding usability patterns. Selected techniques under topics *synchronizing and searching, detecting sequences and comparing sequences* in Hilbert and Redmiles (2000) are evaluated as the basis for the preliminary selection. As the intention is to develop a method that is applicable for different kinds of creation oriented tools, using ideal path comparison of sequences is not possible since there is no knowledge of what the designer is trying to achieve. Instead, for successful detection, using combinations of the detection approaches in this section are considered as a necessary requirement.

Synchronizing and searching of user event data covers automated tools to synchronize and allow the captured data to be analyzed (Hilbert and Redmiles, 2000, p. 395). The topic is also known as a direct usability engineering method type *Capture* that is defined as *"Automatically records usability data e.g., logging interface usage"* (Ivory and Hearst, 2001, p. 473). The available Tekla Structures event log contains both *Windows Presentation Foundation* (WPF) user interface rendering events and *Application-kit* (Akit) events that is Tekla Structures' own user interface event handler. The created log file contains a time-stamp and is therefore useful for multiple techniques that require chronological order.

Hilbert and Redmiles define usability investigator possibilities to two cases where the investigator can either search through UI events or recordings manually for event of interest or check certain point from events or recordings if marked during obser-

vation activities (Hilbert and Redmiles, 2000, p. 396). For this thesis, there is no actual observation phase. Instead, the expert evaluator is able to see the occurrences of user behavioral sequences. As the mining algorithms are searching for repetitive behavior among the captured sequences, the evaluator can focus on inspecting the highest occurrences and categorizing them as occurrence models. After recognizing the repetitive behavior in existing software system, it is possible to do further categorization on whether the pattern is a usability related disruption in normal user detailing tool use process. In Table 7., some typical detailing tool disruptions are listed. On the basis of the captured sequences, some of these disruptions could be accurately detected and transformed into working detection models. In this work, repetitive modify[40] is used as an example for model creation.

Table 7: Sequence example outcomes that are possible scenario for usability problem model creation.

Preliminary examples of found sequence types:
Repetitive search of a design tool
Repetitive tasks inside a design tool
Repetitive modify
Unexpected exit
Unexpected exit and search for similar design tool
Repetitive exit and return to design tool
Deleting created component created by design tool
Exploding created component created by design tool
Transaction delay between events compared to other user data
Error: caused by design tool settings
Error: caused by incorrect pick position in 3D-view
Error: caused by incorrect object pick in 3D-view

Work done by Sanderson and Fisher (1994) deals with useful sequence detection from event related data. The different HCI event types have great significance because they have the tendency to occur in different frequencies (Sanderson and Fisher, 1994, p. 10–13). The extracted usability information from data rich sources that contain a multitude of events can easily produce high amounts of events in short time and be irrelevant to usability analysis. In Fig. 4. Fisher and Sanderson provide good insight to popular HCI related studies and different time related frequencies.

[40]See Section Test application 4.1 for modify definition

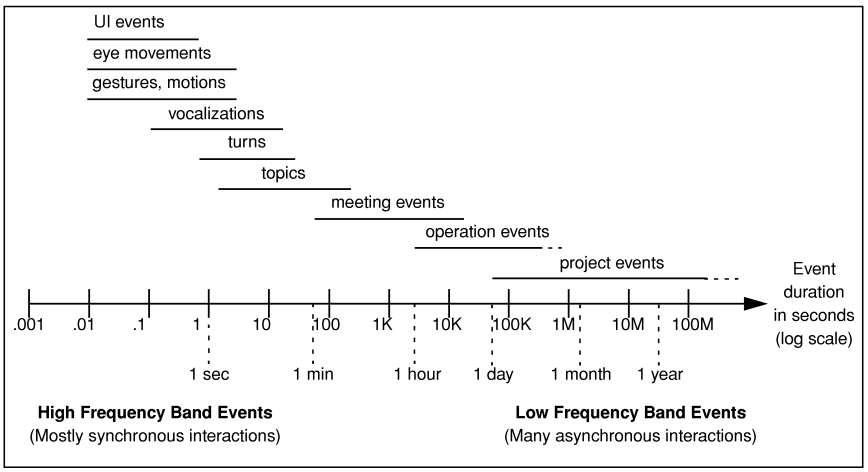


Figure 4: A spectrum of HCI events. Adapted from Sanderson and Fisher (1994, p. 10).

In figure 5., a model of the full event analysis process is displayed. Steps to procedure statistical information from captured user sequences is divided into five different steps that consist of *detection*, *storage*, *analysis*, filtering according to model *boundary condition* and *visualizations*.

Techniques for comparing sequences and their detection for usability related data is targeted for the comparison of source sequences and targets using automation (Hilbert and Redmiles, 2000, p. 386). This is an implementation goal for this thesis, and it is used to automatically extract event patterns. As shown in Fig. 5., the parameters created for each technique can, for instance, be in the form of target sequences that are triggered by key event based rules. Sequence based techniques are also chosen as main type of automated detection for this thesis. Techniques introduced later in this section, such as Fisher's Cycles, LSA or MRPs, are defined as pattern detection type approaches (Hilbert and Redmiles, 2000, p. 404).

The first step into automated problem recognition using repetitive event pattern discovery requires parsing and a sequence detection algorithm that transforms the data into a structured collection of captured user sequences. A SDB is considered to be a constantly growing set of sequence information of all user related specified tasks. In this study, the database contains all detailing tool use sequences regarding macros. After the data is collected and structured, it can be directly analyzed further using selected mining algorithms.[41] It is possible to perform mining according to specific rules or boundary conditions to obtain repetitive pattern information. The boundary conditions are intended to be created using expert evaluators' interest in selected repetitive occurrences seen from data. Table 7. shows an example of potential events of interest. For the test implementation, mapping of repetitive modify boundary conditions were selected to be used during the analysis.

[41] See section 3.3 for selected mining algorithms

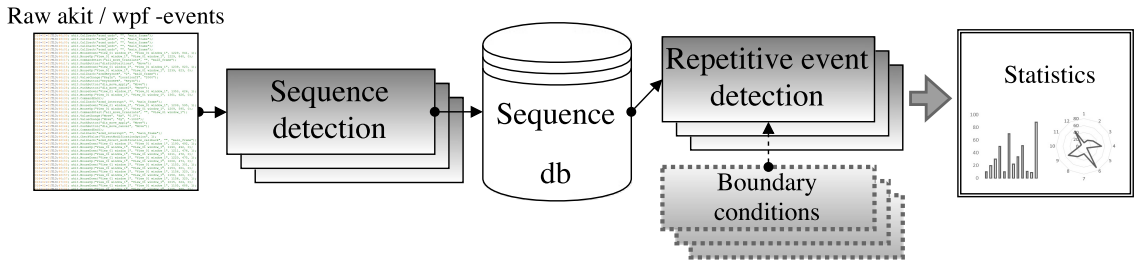


Figure 5: A model of test application analysis process. Based on the models created by experts or examples seen in Table 7., the statistical results can be filtered using given boundary conditions. As a result, the statistics can be used to see occurrences of categorized usability problems types.

Model corresponding sequence types can be transformed into boundary conditions that suit selected analysis approaches shown in Fig. 6. The parametrization of found sequence types can vary from statistical count boundary conditions to short sequences isolated for matching purposes. The found sequence types are categorized under each distinct evaluation method type allowing the parametrization to take place. The working implementation of figure 5. model aims to form abstracted usability related sequence type information into simplified statistical expression. No further user involvement should be required. Depending on evaluator interest, using the implementation can be divided into multiple iterations and depths[42] of interest targeted for the sequence database.

In this thesis, the available approaches are divided into *metric outcome, task model*, and specific *ESDA* techniques that follow some level of pattern matching or logging capabilities. In similar available work, ESDA techniques are considered to be initially used for pattern recognition. Workflow from metric and task models, however, contain multiple principles that could be be used for data filtering. For example, filtering the ESDA created data in respect to specific interest and predefined parametrized statistical count based rules can be used to determine if certain known usability problem types exist.

3.2 Detection techniques in usability research

Available analysis methods were discussed previously in Section 2.4. Inspections included collection of proposed in of computer aided techniques for usability analysis from Ivory and Hearst (2001), Hilbert and Redmiles (2000), Cook and Wolf (1995) and Cuomo (1994). The following approaches were selected based on given conditions in Table 6., and what has been discussed previously.

[42]See depth of recognition in Section 4.2

3.2.1 Fisher's Cycles

Fisher's cycles is a ESDA technique that was introduced in Fisher (1991) by Carolanne Fisher. The technique was a part of Fisher's 'workbench' for implementations of computer-aided protocol analysis. In Fisher's Cycles, evaluators are able to capture user created frequently occurring event patterns using a preselected beginning and ending event that triggers and stops the capturing sequence. The captured event patterns are typically transformed into a summary where the pattern is shown together with the occurrence counts.

Hilbert et al. has provided an in depth example of how Fisher's Cycles would function in an event data sequence. In the example, an investigator is faced with a source sequence of events that are encoded to alphabets (Hilbert and Redmiles, 2000, p. 404–405). In the example, the investigators are only interested in the events that occur between the starting event 'A' and the ending event 'D'. No other sequences are recorded. A sequence can only contain a single starting event. Using Fisher's method a following result occurs:

Table 8: Fisher's cycles example 1. Adapted from Hilbert and Redmiles (2000, p. 405).

Source sequence: ABACDACDBADBCACCCD
Begin event: A
End event: D

Output:

ABACDACDBADBCACCCD
ABACDACDBADBCACCCD
ABACDACDBADBCACCCD
ABACDACDBADBCACCCD

Cycle #	Frequency	Cycle
1	2	ACD
2	1	AD
3	1	ACCCD

From the result, Hilbert et al. note that there were no occurrences of 'B' in any of the recorded cycles (Hilbert and Redmiles, 2000, p. 405). If 'B' were to be considered as a new feature for a BIM detailing tool that is intended to be used between the trigger 'A' and the ending event of 'D', this might implicate a usability problem, as 'B' is not present. Another example better suitable for this thesis might provide more insight. Here we use a similar alphabetic example of multiple users' event data analyzed as one with the following result:

Table 9: Fisher's cycles example 2.

Cycle #	Frequency	Cycle
1	126	ACD
2	6	AD
3	38	ACCD
4	27	ACCCD
5	8	ACCCCD

As we now have a larger number of frequencies recorded, the results with the help of a usability evaluator can provide useful insight. Cycle 'ACD' has the most occurrences and could be confirmed a short and the most favourable work routine performed by users. High amounts of 'ACCCD' or variants might indicate repetitive tasks or problems completing assigned $'A' \rightarrow' D'$ task especially during event 'C'. Like previously, the lack of 'B' in all cycles is interesting and a new cycle implementation with $'A' \rightarrow' B'$ might show that is 'B' resulting in the users retrying of detailing work to achieve $'A' \rightarrow' D'$ goals until 'B' is left unused, and the tasks is completed with 'ACD' instead.

Considering that the summaries are purely user created and follow evaluator interest, the technique is particularly useful for discovering user behavior and workflow in higher frequency[43] tasks without exact parametrized targets. Additionally, if the event data stream provided is structured enough, evaluators could potentially create generalized expectations of the time duration between start and end event, pattern size, or just the failure to reach end event before starting a new evaluation. The sequences inside the recorded pattern are also interesting, and could be further analyzed with parallel Fisher's Cycles implementation or together with other extraction techniques.

In this thesis, some limitations of the Fisher's cycles approach mainly concern capture duration and the ability to follow only higher frequency event streams. Sequence capture can only be implemented to a single detailing tool and the events occurring until component exit or other end of usage. This might lead to a loss of frequently occurring pattern counts due to minimal difference that is irrelevant to actual user task completion. However, this limitation could be partly accompanied by other ESDA approaches, such as MRPs, to connect and capture lower level software usage workflow that interests both usability and software development teams.

In all, additional filtering of sequences obtained in SDB is required, that is, the filtering of unwanted noise from sequences and for the ending itself[44]. Defining the start event type and using options that allow the excluding of certain events defined by the evaluator can help focus on a specific range of HCI event frequency. For BIM

[43] See Fig. 4., the spectrum of HCI events about event frequencies

[44] In this context, it is assumed that the workflow ending is not always trivial, and might require additional filtering and detection effort

software such as Tekla Structures the evaluators could specify multiple different instances of Fisher's for filter events, such as 3D view changes, mouse presses, string inserts, modify and apply events, depending on evaluator interest.

Another requirement for sequence detection in Fisher's approach is the ability to clear the ending, or define multiple event keys for the ending event that stops the capturing of a sequence instead of using one specific key regardless of the real occurred ending event. By using the multiple end key possibility, an equal sequence capture can be interpreted as an equal pattern. A possible implementation could be using data filtering. An example of filtering can be seen in Fig. 6. The use of the technique with posed additions could ultimately lead to many different filtering possibilities to obtain information about tool usage behavior, performance, and even differentiation of locale used as in country requirements and organizations.[45] The focus of development could be in seeing ahead of high priority customer requirements and tailoring specific detailing tools according to the customers' needs by discovering requirements from obtained repetitive data. The implementation of additional filtering is discussed in Section 4.4, regarding a sequence parsing algorithm that creates SDB for mining algorithms.

As in Fig. 6, a similar approach could also be used in parallel. Different Fisher's Cycle parameters could parse through counting discovered sequence definition types, as in Table 7. As the ST, I and ET events are modifiable, the use types and extraction of multiple findings should be possible. The statistical count can be expressed as the summation of basic detailing tool information and the sequence types found overall. In this approach, the discovered 'issues' would be summed and categorized under the detailing tool. This eliminates noise from the available source data and represents information in more useful way. As an example, let us consider that we have discovered one parametrization suitable for Fisher's Cycles with a frequency boundary of 5 that represents *Repetitive tasks inside a tool*. Using this parametrization would categorize all matching cycles' frequencies over 5 together under the design tool name. If the frequencies containing multiple occurrences of 'C' from Table 9. were to represent a similar issue, the start triggered tool would contain 14.6 *Repetitive tasks inside a tool* -issues within the design tool across multiple users, as all cycles with repeated 'C' have the frequency of over 5.

[45] In this context, the locale is test application related country specific settings

```
Start Trigger (ST)= akit.CommandStart*
Igonore (I)       = akit.Mouse*
End Trigger (ET)  = akit.PushButton("saveas",*
                  = akit.PushButton("OK_button",*
                  = akit.Cancel*

ST 9:50:33; akit.CommandStart("ail_create_joint_by_one_sec", "134", "main_frame");
I  9:50:35; akit.MouseDown("View_01 window_1", "View_01 window_1", 747, 651, 1);
I  9:50:35; akit.MouseUp("View_01 window_1", "View_01 window_1", 747, 651, 0);
I  9:50:42; akit.MouseDown("View_01 window_1", "View_01 window_1", 1033, 326, 1);
I  9:50:42; akit.MouseUp("View_01 window_1", "View_01 window_1", 1033, 326, 0);
C  9:50:56; akit.ValueChange("joint_134", "skew", "15");
C  9:51:07; akit.ValueChange("modify_button", "joint_134");
I  9:51:09; akit.MouseDown("View_01 window_1", "View_01 window_1", 1225, 228, 1);
I  9:51:09; akit.MouseUp("View_01 window_1", "View_01 window_1", 1225, 228, 0);
I  9:51:22; akit.TabChange("joint_134", "tw", "jointtab13");
C  9:51:27; akit.ValueChange("joint_134", "edist3", "150");
C  9:51:33; akit.ValueChange("modify_button", "joint_134");
ET 9:52:36; akit.ValueChange("joint_134", "saveas_file", "my_joint");
ET 9:52:37; akit.PushButton("saveas", "joint_134");
ET 9:52:37; akit.PushButton("OK_button", "joint_134");

Output:

ST 9:50:33; akit.CommandStart("ail_create_joint_by_one_sec", "134", "main_frame");
C  9:50:56; akit.ValueChange("joint_134", "skew", "15");
C  9:51:07; akit.ValueChange("modify_button", "joint_134");
C  9:51:27; akit.ValueChange("joint_134", "edist3", "150");
C  9:51:33; akit.ValueChange("modify_button", "joint_134");
ET 9:52:36; akit.ValueChange("joint_134", "saveas_file", "my_joint");
ET 9:52:37; TIMEOUT akit.PushButton("OK_button", "joint_134");

ET time delay = 5s
```

Figure 6: Example of source event data filtering according to proposed additions. Mouse related events are filtered from the capture. The ET event will be recorded as TIMEOUT and is considered as an ending trigger after evaluator given time limit, equal to any other ending types.

3.2.2 Lag Sequential Analysis

Lag Sequential Analysis (LSA) is a technique originally introduced by Sackett et al. (1978) as cited in Cuomo (1994, p. 8). The technique requires a beginning and an ending key event that form the basis for the calculation of the amount of events until reaching the end sequence and the occurrences expressed in sum format. The technique does not show the event sequence itself, but instead expresses the length of events that were required to reach the target. The purpose of this expression is to reveal possible correlation between chosen start and ending event and reveal patterns that might not be strictly sequential (Hilbert and Redmiles, 2000; Cuomo, 1994, p. 405; 8). For example, using Fisher's Cycles requires strict transition between events to be logged as a frequent sequence. In LSA, only the start and ending event remain positioned, and the order of events in various states between them does not affect the final output expression. Due to the possibility of noisy source data, using the technique can reveal otherwise unnoticed correlations between key events (Hilbert and Redmiles, 2000, p. 405).

Hilbert et al. has provided an in depth example of how LSA would function in an event data sequence. In a source sequence equal to the Fisher's Cycles example, the evaluator is again faced with a sequence of events that are encoded to alphabets with starting event 'A' and ending event 'B' (Hilbert and Redmiles, 2000, p. 404–405). Using the technique will output transition lengths between $'A' \rightarrow' D'$ into an expression similar to the following sequence example:

Table 10: Output example of lag sequential analysis adapted from Hilbert and Redmiles (2000, p. 405)

Source sequence: ABACDACDBADBCACCCD
Begin event: A
End event: D
Lag(s): -4 through +4
Output:

Lag	-4	-3	-2	-1	1	2	3	4
Occurrences	0	1	1	1	1	2	0	1

The lag interval chosen for the example in Table 10. is -4 through +4. This means that all sequences up to four events including 'D' will be recorded in the output. The shortest sequence found corresponds to -1 or 1 as the source sequence contains both 'DA', which is negative and 'AD' that is positive traversal. The count one denotes from 'A' or 'D' until reaching target. Lag 2 corresponds to two 'ACD' sequences found and therefore has the occurrence of two (Hilbert and Redmiles, 2000, p. 405).

For this thesis, the use of LSA seemed suitable for assessing the work effort required for each detailing tool in BIM workflow within test application. More cases could be discovered if test implementation was established. The suggestions for the filtering of unwanted events should be available and extended also for LSA. Using large lag intervals could help understand which detailing tools require high changes to various settings during detailing work both using modeling views and the tool interface. Using this information, usability and development teams could evaluate needs to integrate parts of the detailing tool to different existing *Direct Modification*[46] and *floating toolbar* functionalities. Additionally, LSA could be used for problem sequence detection in a way similar to Fisher's Cycles.

Some limitations found of the LSA approach in this thesis concern the recognition of usable patterns. As the technique can provide indirect non-sequential behaving sequences that occur between events of interest, the sequence itself remains unknown unless there is a time-stamp trace relation. Additionally, even if LSA does not require the sequential order of events, noise may still play a significant part in reducing

[46]Direct modification is a Tekla Structures functionality where the user is able to modify certain model objects by dragging handles without using object properties dialog boxes

output accuracy. Obtaining equal lag lengths can provide sequences that have no relation when put to comparison. However, optimal filtering rules and problem sequence parametrization can potentially offer solutions to noise related limitations.

3.2.3 Maximal Repeating Patterns

Maximal Repeating Pattern (MRP) analysis is a another ESDA technique that was introduced in Siochi and Hix (1991). MRP uses an algorithm to parse source event data logging sequences that occur repeatedly (Siochi and Hix, 1991, p. 1). Compared to repeating patterns that occur more than once in a given string, a maximal repeating pattern tries to extract as long a sequence as possible, ignoring smaller substrings (Siochi and Ehrich, 1991, p. 5). However, in Hilbert and Redmiles (2000) MRP also outputs the substrings. In this thesis, both types are used. Siochi and Hix have presented a good example regarding basic output of MRP from a given sequence:

Source sequence: ABCDYABCDXABCE

A simple extraction of MRPs from the sequence outputs 'ABC' and 'ABCD'. The substrings' repeating patterns 'AB', 'BC', 'BCD', and 'CD' are not included since they are part of a longer sequence formation. 'ABC' is a MRP as it represents itself after 'X' (Siochi and Hix, 1991, p. 6). From the perspective of interpreting MRP to recognize all formed patterns, a useful Table expression would output:

Table 11: Example of MRP output collecting all patterns

Pattern #	Frequency	Pattern
1	3	AB
2	3	BC
3	2	CD
4	3	ABC
5	2	BCD
6	2	ABCD

The role of MRPs in usability analysis relies on finding enough repeated sequences to form clear patterns. Usability designers can discover repeatedly occurring real user workflow criteria from MRP output information (Siochi and Hix, 1991; Cuomo, 1994, p. 9; 1–2). The technique does not prioritize any events and is considered unique in the sense that it does not require any evaluator defined starting or end target definitions (Cuomo, 1994, p. 10).

Challenges in the usage of MRPs for BIM event data are related to transcript file amounts. MRP can easily create a large number of transcripts that contain multiple frequencies. Most of the transcripts can be considered to be noise, and they typically do not indicate usability or workflow related patterns (Siochi, 1989,

p. 59–60). However, the amount of transcripts follows a linear scale[47] in number of event lines in respect to number of MRPs detected. As the number of detected MRPs rises, the number of frequencies of longer patterns is also revealed[48] (Siochi, 1989, p. 59–60). Using filtering rules based on specific interest could possibly eliminate noise from MRP transcripts. Evaluator defined parameters and concentrating on given log time interval pattern length could improve transcript importance.

The use of MRP for usability problem recognition as well as the discovery of user overall workflow is considered highly useful for this thesis. Similar to Fisher's Cycles, the use of filtering of evaluator defined events should be available to make full use of the technique. Using strict filtering, such as excluding all event types apart from e.g. figure 6. 'akit.CommandStart*', can create interesting samples of user design behavior in longer time periods regarding lower frequency usability events such as *operation events* defined in Fig. 4., where a single event can occur in a timeframe of hours up to days. As the events also contain time-stamps, there is the possibility of logging detailing tool transformation periods for analysis. Detailed searches directed towards these transformation periods could indicate the use of external design tools for some parts of the workflow. From a pure usability related perspective, using MRP to find frequently occurring sequences without any predefined interest can reveal usability related problem patterns in large event data collections. The data does not need to cover solely design tool related filtering, but instead 'akit.*' or 'wpf.*' events of the overall software user interface. Accessing the sum output of MRP and seeing the highest occurrences should be the focus of the usability teams' interest for further analysis.

3.3 Selection of mining algorithms

After inspecting and selecting suitable approaches from the usability domain, a comparative search was conducted into the modern data-analysis domain. The search for suitable analysis methods was limited to SDA. This allowed a sufficiently narrow inspection of open source *Frequent Itemset Mining, Sequential Pattern Mining*, and additional *Sequence Prediction* approaches. Available methods are a part of the collection in SPMF, an open-source data mining library in Fournier-Viger et al. (2014b).

Before choosing each individual method for further analysis, tests were conducted to compare each algorithms *performance* and *compability* against test application captured random sequences. Based on the inspection, BIDE+ and TKS were the fastest[49] and had the best equivalence to FC and MRP. Additionally, as discussed in usability related approaches in Section. 2.4, a suitable pattern prediction algorithm was found. CPT+ is a useful learning based approach identifying the most likely pattern of specified user tasks if SDB was filtered for single detailing tool.

[47] See Fig. 10. in Siochi (1989)
[48] See Fig. 12. in Siochi (1989)
[49] Based on comparison to test application data-set, see Section 5

By testing and reviewing theory on each inspected mining approach, frequent itemset mining was found incompatible for the test application data. The test application data consists of items in a transaction that may appear more than once per transaction. The limitation of the frequent itemset approach is that the items are allowed to appear only once in a given transaction (Fournier-Viger et al., 2014b).

Based on the test findings, there was a moderate to high amount of variance between sequences. Using *non-closed* pattern mining such as PrefixSpan resulted in high number of found patterns. To limit the amount of found patterns, I chose to use *closed sequential pattern mining*[50] approaches. Also, in Wang and Han (2004) regarding BIDE+, it is noted that using a closed approach resulted in more compact, complete and efficient patterns (Wang and Han, 2004, p. 1).

Table 12: Inspected mining approaches for data analysis. Consists of SPMF library (Fournier-Viger et al., 2014b). Based on the inspections, BIDE+, TKS and CPT+ are the most suitable algorithms for the test implementations.

Method	**Algorithm**
Frequent Itemset	
	FPGrowth
	DCI_Closed
Sequential Pattern	
	PrefixSpan
	CM-SPADE
	ClaSP
	CM-ClaSP
	CloSpan
	BIDE+
	TKS
Sequence Prediction	
	CPT+

A direct approach for LSA was not found and was therefore excluded from the test implementations. Utilizing LSA for time constraint based detection is however still considered as a valuable approach to implement if suitable application is found.

3.3.1 BIDE+

BI-Directional Extension based frequent closed sequence mining (BIDE) is an algorithm for discovering complete sets of frequent closed sequences. The algorithm's efficiency is based on the fact that there is no direct requirement for maintaining a large amount of possible pattern candidates (Wang and Han, 2004, p. 1–3). By using the BIDE *pattern closure mechanicsm*, consumption of memory and running

[50] *Frequent closed sequence* e.g. s_a can be defined as *closed* if there is no other larger sequential pattern s_b, where s_a is a subset and has the same support (Fournier-Viger et al., 2014a, p. 42)

time is lower than CloSpan, SPADE or PrefixSpan[51] (Wang and Han, 2004, 2). Bide was also the fastest algorithm for the test application example data-set.

A good definition of BIDE+ input and output comes from Fournier-Viger et al. (2014b). Input sequence format can contain items and or itemsets. Due to the input format, items are presented in integer format, but follow the same guidelines as the previously introduced FC, LSA, and MRP. A sequence could contain multiple itemsets such as $s_1 = \langle\{1,2\},\{1\},\{4\},\{1,2,3\}\rangle$ or single items $s_2 = \langle\{1\},\{2\},\{4\},\{3\}\rangle$, following also the same definition as in Section 2.2.3. For test application data, all implementation follows a single item format similar to s_2. In addition, all items or itemsets in a sequence must be assumed to be lexically ordered (Wang and Han, 2004, p. 3). Although it's not a constraint for test application data, it should be remembered that patterns categorized under sequential pattern mining in SPMF generally restrict that an itemset, such as s_1, cannot contain duplicate values (Fournier-Viger et al., 2014b).

Table 13: A single item sequential database example with duplicate items in a sequence. The example is similar to the ones found in captured event sequences in test application.

$$SDB = \begin{cases} s_1 = \langle\{1\},\{2\},\{3\},\{5\}\rangle \\ s_2 = \langle\{1\},\{4\},\{5\},\{4\}\rangle \\ s_3 = \langle\{1\},\{3\},\{2\},\{3\},\{5\}\rangle \\ s_4 = \langle\{1\},\{2\},\{3\},\{2\},\{5\}\rangle \\ s_5 = \langle\{1\},\{2\},\{3\}\rangle \end{cases}$$

Using example SDB in Table 13. outputs similar resulting data as with FC shown in Table 8. or 9., by sequence comparison. Compared to non-closed, the pattern yield using the same minimum support (minSup) is 19 patterns. The algorithm in SPMF library also has the ability to set maximum pattern length. The initial use cases of BIDE+ are similar to those defined for FC in Section 3.2.1. From the test application viewpoint, the patterns reveal frequent behavior and links among the user workflow. The example resulting cases from the test application are shown in Section 5.

[51] See CloSpan, SPADE or PrefixSpan in Fournier-Viger et al. (2014b, 8–11)

Table 14: BIDE+ outputs frequent closed patterns using minSup. In this case pattern five has the minSup of 40%.

Pattern #	Support	Pattern
1	5	$\langle\{1\}\rangle$
2	4	$\langle\{1\},\{2\},\{3\}\rangle$
3	4	$\langle\{1\},\{5\}\rangle$
4	3	$\langle\{1\},\{2\},\{3\},\{5\}\rangle$
5	2	$\langle\{1\},\{3\},\{2\},\{5\}\rangle$

3.3.2 Top-K Sequential pattern mining

Top-K Sequential Pattern mining (TKS) is an algorithm for mining user defined k sequential patterns in given set L and sequence database SDB. The top-k mining results in maximum support sequential patterns (Fournier-Viger et al., 2013, p. 1–3). If each pattern $s_a \in L$, there is no sequential pattern $s_b \notin L \mid sup(s_b) > sup(s_a)$ (Fournier-Viger et al., 2013, p. 2).

The algorithm was initially created to address difficulty of setting minSup values. The problem is analogous to other related studies[52] of frequent itemset, association rule and sequential rule mining variants (Fournier-Viger et al., 2013, p. 4). Similar approaches exist, such as Top-K closed sequential pattern (TSP) introduced in Tzvetkov et al. (2005). TKS was created to be faster and more memory efficient, particularly when using dense SDBs (Fournier-Viger et al., 2013, p. 2, p. 10–12).

The SDB example in Table 13. outputs similar data as the MRP shown in Table 11. In addition to selecting value k, the TKS implementation contains user-selectable options *minimum pattern length, maximum pattern length, required items* and *maximum gap between items*. Required items allow the specifying of items that must appear in each found pattern. Maximum gap allows a gap interval between pattern contents. If gap is set to N, a gap of $N-1$ itemsets are allowed between the previous and following itemsets of a pattern (Fournier-Viger et al., 2014b, TKS Documentation). For example, if we run TKS to SDB using $k = 5$, minimum pattern length of 3 items and required items using item 1, the top-5 most frequent patterns are:

[52]See Wang et al. (2005) for frequent closed itemset, Fournier-Viger et al. (2014b, Top-K Association Rules) for association rule and Fournier-Viger et al. (2014b, Top-K Sequential Rules) for sequential rule top-k examples

Table 15: TKS outputs top-k five patterns. TKS calculates minSup in preprocessing and after exploring the possible pattern candidates. In this case, final minSup was two, revealing six individual patterns that have the support of over two.

Pattern #	Support	Pattern
1	4	$\langle\{1\},\{2\},\{3\}\rangle$
2	3	$\langle\{1\},\{2\},\{3\},\{5\}\rangle$
3	3	$\langle\{1\},\{2\},\{5\}\rangle$
4	3	$\langle\{1\},\{3\},\{5\}\rangle$
5	2	$\langle\{1\},\{3\},\{2\},\{5\}\rangle$
6	2	$\langle\{1\},\{3\},\{2\}\rangle$

3.3.3 Compact Prediction Tree+

Compact Prediction Tree (CPT+) is a lossless or near lossless[53] tree structure based sequence prediction model that predicts next item of a given pattern (Gueniche et al., 2015, p. 2–4). CPT differs from other existing models such as Prediction by Partial Matching (PPM) or markovian, All-K-Order Markov (AKOM)[54] in that it utilizes training sequence information more efficiently. The model can be used to predict the next items of subsequences that have not been previously introduced during sequence training. As a result, CPT is able to predict the next items of subsequences with noisy data (Gueniche et al., 2013, 2015, p. 1–2, 3–4).

The model approach is divided into two main phases. First, the sequences are trained and compressed from input SDB forming a sequence prediction model. For CPT+, this work uses Frequent subsequence compression (FSC-strategy). This is based on using a modified[55] PrefixSpan algorithm to identify all sequential patterns and the insertion of these to the CPT.

The basic insertion and training consists of a *Prediction Tree* (PT), *Inverted Index* (II) and a *Lookup Table* (LT) (Gueniche et al., 2013, p. 3–4). Each sequential pattern training sequence is inserted to the PT one after another. If the root in a PT has a direct child, the sequence will follow existing paths. If there are no direct matches from root, a new child is inserted with the item's value. The inverted index contains information of the sequence in which a given item appears, and stores this information for further lookup to find sequences containing a set of items (Gueniche et al., 2013, p. 3–4). PT and II are linked together using the lookup table. For each sequence, LT points to the last item in PT (Gueniche et al., 2013, p. 3–4). As an example, we can look at a CPT construction of the following two sequential patterns:

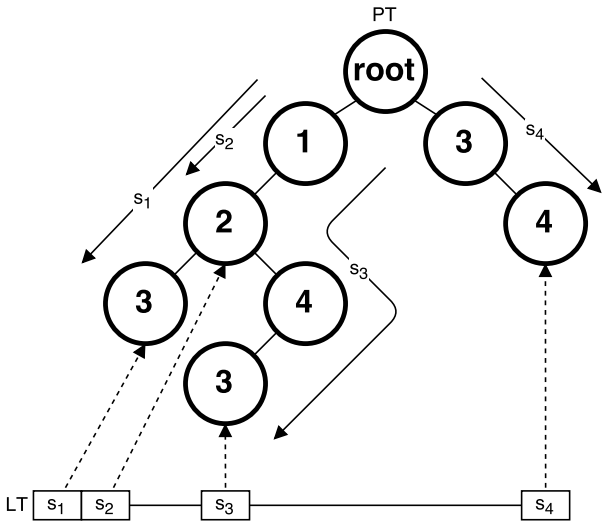
[53] In this context, loss of information in training sequences is dependant on the kind of data compression method used

[54] See PPM or AKOM in Fournier-Viger et al. (2014b)

[55] Modification limits found subsequent itemsets between assigned size constraint (Fournier-Viger et al., 2014b, CPT+ source code)

Table 16: Found sequences to be inserted to a PT

$s_1 = \langle\{1\},\{2\},\{3\}\rangle$
$s_2 = \langle\{1\},\{2\}\rangle$
$s_3 = \langle\{1\},\{2\},\{4\},\{3\}\rangle$
$s_4 = \langle\{3\},\{4\}\rangle$



Item	s1	s2	s3	s4
1	1	1	1	0
2	1	1	1	0
3	1	0	1	1
4	0	0	1	1

Figure 7: Prediction tree and lookup table (right) and Inverted Index (left) constructed from given sequential patterns from Table. 16. above. The pointing arrows represent the pattern traversal.

Predicting the next item is done in the next phase by comparing a sequence against trained sequences (Gueniche et al., 2015, p. 3–4). If x represents an integer of prefix length[56], making a prediction for sequence s is created by finding all sequences that contain the last x items for s in any order or position (Gueniche et al., 2013, p. 5). The accuracy of prediction is dependant on the prefix length and data-size. CPT is showing the highest accuracy using a prefix pattern of eight (Gueniche et al., 2013, p. 10). The found sequences are called *sequences similar* to s and used to predict the next item of s(Gueniche et al., 2013, p. 5).

When looking at the created PT, II and LT in Fig. 16., a prediction can be constructed for a given sequence. For example, to predict the next item of pattern p_1 $\langle\{1\},\{2\}\rangle$, we can start by looking at the II items 1 and 2. Common sequences are present in the index and found from s_1, s_2 and s_3. By using backwards traversal in the LT, it is possible to count common occurrences after the given prefix. For this, the procedure uses an additional *Count Table*. For pattern p_1 the item 3 has a higher count of two, and is presented as a next item prediction.

[56]In this context, the given items for predictor

In this work, CPT+, an improved version of CPT, is used. Compared to other prediction approaches[57], CPT has a higher order of spatial complexity[58] to output predictions. CPT+ was developed in a paper by Gueniche et al. (2015) to reduce higher prediction times by proposing two new compression strategies for subsequences and tree branches, and a noise reduction strategy (Gueniche et al., 2015, p. 2)

For the test application data, the use of noise tolerant prediction was considered appropriate. The application pattern lengths were moderate, and from the CPT limitations' point of view, decreased accuracy after prefix length of eight did not present a direct issue. Average pattern lengths in test application data can bee seen in Section 4.3.

[57]See PPM, DG or AKOM in Fournier-Viger et al. (2014b)

[58]In this context, spatial complexity is defined as the order of magnitude that is required for working storage to solve initial problem

4 Implementation of methods

This section explains the implementation done for the study and examines the automated analysis procedures suitable for test application. First the test application and it's basic functionalities are introduced. The section then continues introducing working hypothesis of software use behavior and the depth of information required to be captured for the given research questions. The latter parts of the section discuss the usability test setup process, problem identification, and the automated detection of sequences. The problem identification and automated detection follows the given hypothesis. The end of the section shows sequential mining and prediction use process model applied to the test data-set outside of usability study definitions, described previously in section 3.3.

4.1 Test application

The selected BIM software for the test application is Tekla Structures. Tekla Structures is a BIM software specialized in structural design. By using the software, structural engineers are capable of delivering designs for construction and built environment projects. The projects created by users vary from residential buildings through plants and factories to bridges and offshore structures.

Detailing tasks done in the software are typically executed by structural engineers. The detailer works in a 3D environment that consists of structural solid modeling objects in categories such as *columns, beams, plates, panels* or *slabs.* Objects are created in the model and later detailed according to design and structural needs. Detailing can be achieved either by manually editing or using automated applications and components that perform pre-defined detailing routines and logic.

In manual detailing, the model object's geometry is modified by using cut objects or additional objects that are added to existing objects as subobjects. Manual detailing does not follow any constraints that might exist using automated tools. Common categories of objects added as subobjects are *plates, bolts* and *concrete reinforcing rebars.* subobjects and main modeling objects often form a single *assembly*[59] or *cast unit*[60]

Automated detailing is based on existing knowledge of structural details. Routines are introduced to selected objects forming structural details. The tasks may include the creation of *custom parts, connections, details* or *seams.* Part of Tekla Structures development is focused on the delivery of components and applications that are useful for different user groups. Depending on the specifications, tools are also developed to suit global or specific area needs.

[59] Several steel modeling objects are joined together forming a build assembly

[60] Several concrete or reinforcing parts joined together ready to be casted as a whole

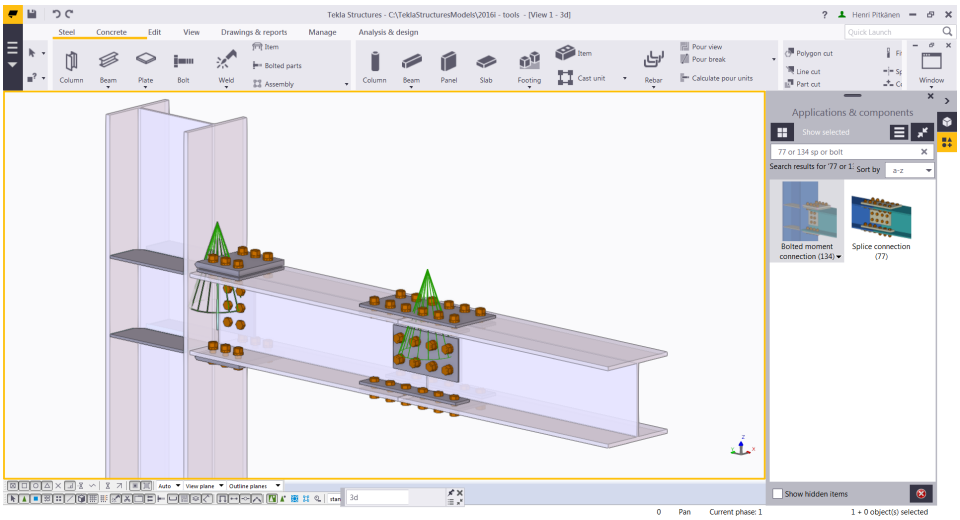


Figure 8: Test application user interface steel detailing example. Detailing tools *Bolted moment connection* (134) and *Splice connection* (77) are applied to modeling objects and seen in gray and brown with visible green cone shaped *Component symbol.* After creating the connections, the three individual steel parts that consist of a column and two separate steel beams form a single assembly.

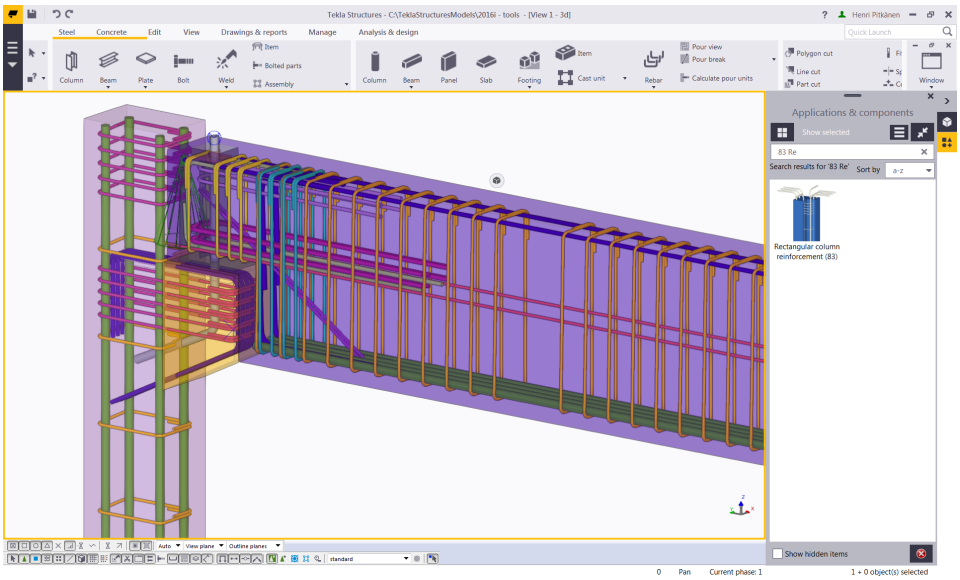


Figure 9: Test application user interface, concrete detailing example. Detailing tools *Rectangular column reinforcement* (83), *Corbel connection* (14) *Corbel reinforcement* (81), *Beam end reinforcement* (79) and *Beam reinforcement* (61) have been used. The corbel is a concrete part that is added to a column as a subobject forming a cast unit. The reinforcing tools' routine-created rebar modeling objects are added as subobjects to each main modeling object. Due to the predefined settings of the detailing tools, the beam and its reinforcing are separate cast units.

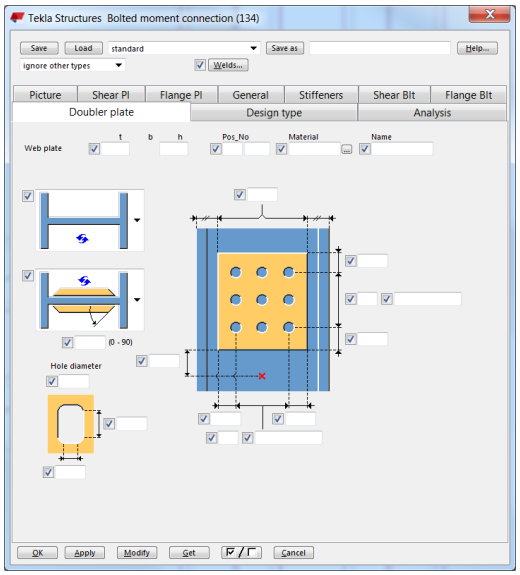


Figure 10: System Detailing tool Bolted moment connection (134) tool Doubler plate tab. In this tab structure, the user is able to change bolt plate dimensions and bolt hole configuration using various text boxes that take integer values as input. The user is also able to change plate material information and displayed name information. The tool requires two steel I-shaped beams as input modeling objects in order to create default detail.

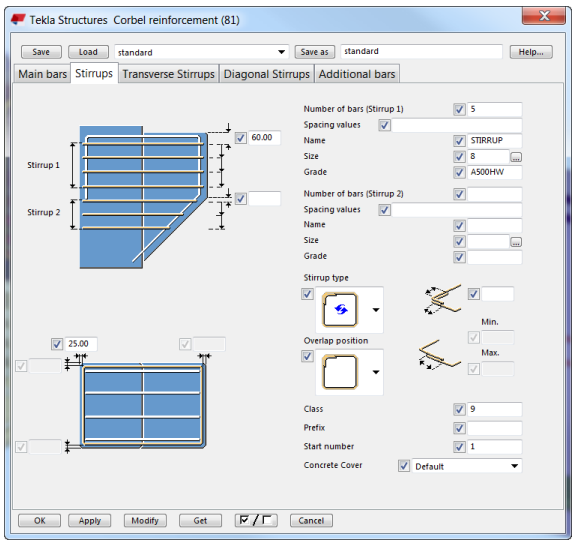


Figure 11: System detailing tool Corbel reinforcement (81) Stirrups tab. In this tab structure, the user is able to change reinforcing stirrup positioning, amount, size and overlap types. The user can also change stirrup material information and some additional values, such as reinforcement numbering starting index. The tool requires a column as a secondary input in addition to corbel due to the reinforcing being extended to the column and being a part of the same cast unit.

In figures 8. and 9., the model view is shown on the left. The view shows the main modeling objects and applied detailing. The main modeling objects are created by accessing ribbon in the upper part of the interface. The detailing tool side panel[61] is open and showing a filtered view of the used tools on the left. The dialog of a detailing tool can be accessed from the model view by either double-clicking connection, its green symbol, or directly from the tool thumbnail. In the dialog structure the user can modify, apply or save settings, or seek help. The detailing tools used in the figures automatically create structural connection or reinforcing based on default or user defined settings and modeling object geometry. The connection parts created by the detailing tool are added to the modeling parts as subobjects according to predefined rules.

[61] The side panel containing detailing tools in the test application is called *Applications and components*

In order to create a basis for recognition, there is a need for an initial working hypothesis base for software use behavior in BIM detailing tool use. For the test application, there was no previous research available. A common use case of a detailing tool workflow is considered to be the following.

User workflow order is expected to consist of the creation of modeling objects and later applying detailing tools. The detailing is commonly divided into two steps. The user applies the detailing tool from the *applications and components* window or side panel with default or saved settings. Applying the detail tool in the model functions by selecting each modeling object as input with the mouse. After viewing the results in model view, the user then decides if the selected tool representation is correct and suitable for solving the given structural design task. The next step is to open the detailing tool settings dialog and change the necessary parameters to create actual structural detail according to the exact design. Changing the parameters and modifying the changes iteratively is also common. If the used tool is not fully familiar to the user, partial changes and seeing the model changes visually help the user assess if the detail is correct. The detailing task commonly ends after the creation of preferred and correct detail. The user's individual detailing task can be assumed to end after using the detailing tool interface *OK, Apply, Cancel, Save as* or standard *form close* -command combinations. These commands can be seen as interface buttons both in figures 10. and 11. In rarer cases, users can leave the detailing tool open without leaving an exact trace of ending a detailing task. In these cases, the end is determined by the last specific tool related command before re-activation of the same tool, or starting a new detailing task with another tool.

4.2 Depth of recognition

By using the proposed methods of analysing event data, it is possible to pinpoint areas of repetitive behavior that the user is facing without human interaction. The test implementation is able to construct such data from the user event stream without any disruption or the user even being aware. The chosen methods and parameters in this solution are required to be applicable to such an extent that the individual configurations for each tool is not required. Example generalizations can be divided into three levels of depth.

1. The user is not able to find the best suitable tools to perform required task.
2. The task that the user is trying to achieve has disruptions.
3. A certain element within the user task is causing disruption

The first depth *The user is not able to find the best suitable tools to perform required task* is for receiving data about user search keywords and trending searches. This could be used to obtain certain user segments' typical search entries and into which tool these searches ultimately lead and whether the component works for used objects. The level could include the recognition of the interval between search and component launch until possible exit without using a component. Additionally, the

number of occurring searches in a short period of time until the component launch could be included in this level. By using statistics created with these patterns, there is a possibility of automatic discovery in situations where users are having difficulties finding the correct tools. Searches could also indicate desired functionalities for future software development.

The second depth of recognition *The task that the user is trying to achieve has disruptions* could allow the usability team to minimize their search of potential issues and prioritize the tasks that have the highest statistical occurrence. This is a form of pattern detection that identifies occurrences with concretely or abstractly defined parameters in collected event stream (Hilbert and Redmiles, 2000, 404). Using construction from event pattern recognition elements could reveal a single tool or toolset that causes usability breakdowns. The prerequisites for this analysis type require the creation of generalized and expected behavioral models as the basis for parameter definitions (Akers et al., 2012, p. 45).

The third level *A certain element within the user task is causing disruption* focuses on recorded data inside a certain task. This could decrease issues in finding the exact location of a disruption from a generalized level to an area in used component structure. The discovery of the disruption could possibly be generated using event patterns that evaluate component dialog exit and time usage.

4.3 Available data

Existing user event data from the test application was readily available. The data has been collected since the Tekla Structures version 20.0 from users who have accepted to participate in the Tekla Structures UFP-program. Not until the version of 2016 a portion of the log files has been lost due to asending process that involves data sending interval triggered by a visible prompt that can be canceled. The feedback program has since developed and transferring of the data is now a background process. The data is gathered to an external server and saved in zipped *.txt* containers. User feedback log files are being sent to the server from each user after a default of 10000 lines of interaction. Data is pushed to the server after each full line-count, allowing continuous data fetching. Possibility of utlizing live-data is discussed in section 6., as future implementation of continuous sequence detection.

During the inspection of the test application data done for this paper, three known event loss causes were identified. Primary cause was found to be the large reform of the test application of the 2016-version UI layout and utilizing the new WPF-technology. A part of the user actions are not mapped, and they are missing from the current event data log. The use of all UI actions is possible, but results may contain a loss of events. All sequences under analysis in this paper are targeted towards tools that have mappings idenfified functional.

Secondary cause found was the missing UI mappings of plugins and applications. Other than registering *CommandStart*-of each plugin or application, there was no other log data found from .dll-based plugins or executable applications.

Other causes of data loss were found to be related to test application parts that have been under lesser analysis interest. Tekla Structures Drawing-base[62] contains a higher amount of missing mappings. Requirements related to missing valuable mappings are further discussed in Section 6.3.

The test analysis data consisted of the Tekla Structures version 2016 repository containing 26.3GB set of user recordings with 555534 log files. The files did not limit to the 2016 version logged data, but instead represent the current version timeline during retrieval. Macro related detailing tool sequence detection and the creation of a test *SDB* yielded 41173 sequence starts and 12668 individual user-sequences containing user actions. The total number of individual user actions recorded were limited to detailing tool UI actions. The filtering output of total user actions in *SDB* was 219529. Additional data processing operations that were executed regarded UI noise, divided into two separate cases. First, fetching only specific tool sequences from *SDB*. Second, filtering high occurences of known, generic Ok, Apply, Modify, Cancel, or tab change *"tw"*-commands. If there was no interest in inspecting high occurence of transfers between detailing tool and test application model view, additional filtering might apply. The use of additional filtering is mentioned in all test analysis result cases if used.

Based on the inspection of existing data format and creating a test implementation that parses the data, the following technical and process depth related factors were taken into consideration.

Operating system character formatting:

Varying formatting type could result in fetched file read operations to fail. Using formatting to change function for each line that is being read is recommended. As a drawback, this might slightly affect parsing time.

Operating system locale:

As the event data obtained is not from any specific region, using different locale could result in variation in timestamps. The detection and parsing of used standard date and time notation formatting is recommended. The sequence detection process uses time extensively for positioning and calculating last sequence event instance during recursion. Additionally, the timestamps are used to obtain other time-related usage information if so required.

Command positioning:

[62] In this context, test application has model-view and a drawing-view for creation of documentation

While parsing lines, the entry is being processed into a list of basic elements consisting of a *Timestamp (time), Command Name (cmd), command-id (cmdId), Target* and a *Attribute field (attrib).* The existing raw data often requires change between logical commandname and cmdId. As an example, starting macro detailing and pressing modify in the tool dialog will trigger the following log entries:

```
ST 2.1.2016 14:26:01; akit.CommandStart("ail_create_macro", "30000079", …
C  2.1.2016 14:26:20; akit.PushButton("modify_button", "macro_30000079");
```

Figure 12: Example of unstructured data sequence detection start trigger "akit.CommandStart("ail_create_macro"" and first dialog structure edit command "akit.PushButton("modify_button"".

The commandname "ail_create_macro" or "modify_button" is not as specific as "30000079" and "macro_30000079". During sequence detection, it is necessary to have a specific identifier where the design toll can be traced as a sequence. For this purpose, in the above example, the command-id is a correct identifier. For the rest of the sequence, however it must be noted in what level of depth information is gathered.

The second depth of recognition *The task that the user is trying to achieve has disruptions* has a requirement for inspecting detailing tools and the general actions that were used, but it does not specify the exact location in the dialog structure. For the most part, the used general actions are structures that have globally equal formatting and are offered as pre-made structures for detailing tool development. In Table 17., the second depth of recognition requires *cmd* column as minimal requirement for available SDA techniques.

In the third depth of recognition *A certain element within the user task is causing disruption* the available data requires additional comparison elements, such as using the specified field in this case. Detailing tool development does not follow strict guideline for form item naming outside of premade interface structures. This limits the possibility of detailing tool comparison. The third depth compares individual tool usage and recognizes exact paths among captured sequences.

Table 17: Example of captured and parsed sequence of detailing tool macro 80000006. A *sequence Id (sId), Sequence Position (sPos)* is manually added to the existing data. The cmd field is a combination of cmd and cmdId columns. sPos "1" cmd is from cmdId and the rest of the is native cmd from sequence. Using a combination of the *target* and *attrib* tables will reveal detailed information of the used tool. If using target, the evaluator is able to see exact interface form item used. By using the attribute column, it is also possible to extract the user input settings of individual tools.

sId	**sPos**	**time**	**cmd**	*target*	*attrib*
13d0812d	1	2016-01-29 14:26:01	80000006		
13d0812d	2	2016-01-29 14:26:03	akit.TabChange	*tw*	
13d0812d	3	2016-01-29 14:26:09	akit.ValueChange	*type*	*test*
13d0812d	4	2016-01-29 14:26:10	akit.TabChange	*tw*	
13d0812d	5	2016-01-29 14:26:30	filename_btn		
13d0812d	6	2016-01-29 14:26:36	akit.ValueChange	*dens*	*test1*
13d0812d	7	2016-01-29 14:26:44	apply_button		
13d0812d	8	2016-01-29 14:26:47	akit.TabChange	*tw*	
13d0812d	9	2016-01-29 14:26:56	akit.ValueChange	*sname*	*test2*
13d0812d	10	2016-01-29 14:27:00	akit.ValueChange	*pname*	*test3*
13d0812d	11	2016-01-29 14:27:00	apply_button		
13d0812d	12	2016-01-29 14:27:09	OK_button		

The first depth of recognition: *The user is not able to find the best suitable tools to perform required task* is a separate detection case in comparison with the second and third depths that follow a fairly similar approach. The available data has a clear detectable structure that contains user search input as shown in figure 13. The initial detection is triggered by this search structure and ends when the user finds the correct detailing tool. The correct tool can be found when the user stops repetitive tool opening behavior and or starts a detailing tool editing sequence.

```
ST 13:45:46; wpf.View("… Find … SetText("column");
C  13:45:52; wpf.View("… Find … SetText("concrete column");
C  13:46:01; wpf.View("… Find … SetText("column shoe");
ET 13:46:08; akit.CommandStart("ail_create_plugin","ColumnShoeConnection", …
```

Figure 13: Example of detecting first depth recognition. The ST and C follow a clear structure. The ending trigger requires additional constraint based detection.

4.4 Test implementation

Automated sequence detection and predictions were made as a part of the testing implementation. The used data was real user event data. The detection consisted of the following phases: 1) *Working hyphotesis*, 2) *Sequence detection*, 3) *Transformation and processing of data*, 4) *Sequence mining and prediction*. Foundations of the above phases are introduced in this section.

Sequence detection from available data[63] has been created to suit repetitive pattern discovery requirements. For this work a An Open-Source Data Mining Library by Philippe Fournier-Viger et al.[64] was used with detected sequences for validating purposes. The sequence detection algorithm was made for this thesis work and is suitable for test application event data.

Primary detection from the existing event data stream is based upon *Start Trigger* and the optional *End Trigger* based on MRPs and Fisher's requirement, as shown in the preliminary example of event data triggering in Fig. 6. Detection also uses *ET time delay* expressed to allow recursive or fixed time window searches for sequence completion and *Ignore* functionality to filter unwanted existing noise. For macro detection, raw data contains an identifier sId for each line of action. In this case, there is no need for end trigger, but recursive checking for parallel detailing tasks[65]. Detection used in this work follows the approach shown in figure 14. In this example, A is the available raw data being read. A_i is the line of raw data that is being inspected. *trigger* is a list of trigger keywords defined by the user. *sequence* is the position in a sequence. *queue* is used when an unexpected *trigger* is found during a sequence capture. After handling the current sequence until end of the file or found $endTrig$, items in queue are recursively handled and sequences added to the SDB in the correct position. *start* is a list of used timestamped triggers. Parse(A_i) modifies single A_i to a list format[66] $endTrig$ is a known ending cmd. *ignore* is a list of ignored content defined by the user. Time($SeqId$) translates to a sequence start timestamp, whereas t is the maximum used ET time delay to continue recursive search if no ending has been found. The capture sequences are transferred to a SQL server database for storage and transformation operations for future analysis implementation.

After detecting the sequences from test application data and pushing them into a database, the data was transformed into sequence and prediction approach formats. Transformations were done directly in a SQL database as table transformations using hash matching to give frequent values a common identifier. The used SPMF format is a transposed integer format of given sequence database contents, shown

[63] See section 4.3. Available data from software

[64] See library in http://www.philippe-fournier-viger.com/spmf/, or the related paper Fournier-Viger et al. (2014b)

[65] In this context, when 1) user starts one detailing tool and 2) opens another without finalizing the first task. In recursive check, both tasks are recorded and ordered in the SDB starting from the first task

[66] See Table 17. for example formatting

```
function ANALYSISFILE(A, startList, SDB, queue)
    sequence ← 0
    for A_i in A do
        if A_i is trigger then
            if A_i in queue and sequence > 1 then
                Add A_i to queue
            else if A_i not in start then
                sequence ++
                Parse(A_i)
                Add A_i to (start, SDB)
                SeqId = A_i[sId]
        else
            if A_i not in ignore or sequence ≥ 1 then
                if Time(A_i) − Time(SeqId) > t or A_i is endTrig then
                    if items in queue then
                        queue.pop
                        Analysisfile(A, startList, SDB, queue)
                    return
            else if Valid for (SeqId, A_i) then
                sequence ++
                Parse(A_i)
                Add A_i to SDB
```

Figure 14: Simplified representation of macro sequence detection used for test implementation. The approach reads multiple log files averaging 1140 KB in size.

for example in Table 17. From test SDB, column *cmd* was being used for sequential pattern and pattern prediction.

Data processing involved certain procedures regarding the created SDB. Based on interest, the data contents were filtered to a smaller size, containing specific detailing tools only and or removing generic actions for reduced noise. All data was fetched as an extraction of the SDB data in SPMF converted .txt format. Processing modifications were done to the extracted data, leaving the SDB to its original state.

The SPMF format was directly compatible with all of the selected mining and prediction approaches. The performance varied depending on the processing of data and used options[67] The output format consisted of SPMF integers that were translated back to traceable test application events using an indexed translation list. Additionally, the tool name translations were maintained in a separate listing and

[67]See algorithm mining or compression options in Section 3.3

required full translation. The translations of the *cmd* commands are further usable as they are tool specific UI button identifiers.

The method implementation process followed the path shown in Figure 15. The sequence detection and initial filtering was done in *SequenceDetect* during parsing of the data as first steps. The test implementation data used ST *Trigger* "akit.CommandStart _create_macro". Additional user based filtering was done after *SDB* push. Only outputs from *SDB* are further used for sequential mining or prediction. The *SequenceDetect* is only run to newly acquired raw data.

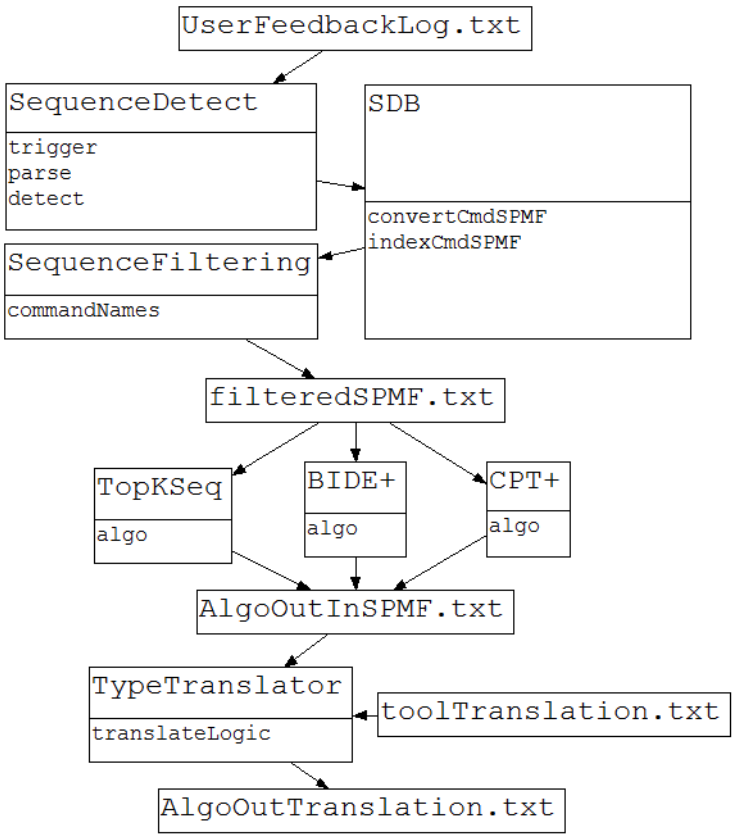


Figure 15: A simplified process chart of method implementation, starting from raw user log data. The SequenceDetect parser contained filtering that limited the detection to macro-detailing tools. SequenceFiltering was used to specify additional filtering, such as the removal of noisy action or evaluation of a specific tool. The algorithms TopKSeq, BIDE+ and CPT+ were compiled from SPMF source code as individual console .jar-applications, reading filtered SPMF.txt-files initiated by simple control script. The *SDB* was controlled by a SQL-server. Other process parts were done as Python implementations, including the actual sequence detection.

5 Results

This section discusses the example cases and visualizations using either the test SDB (See examples 5.2, 5.3 and 5.4) or a combination of BIDE+ and TKS for pattern discovery (See examples 5.1 and 5.5). The purpose of these analysis methods are to experiment with the available approaches and demonstrate how sequential data or ESDA approaches data could be utilized in the contemporary BIM domain. Towards the end of the section, the use of sequential pattern data for pattern training and prediction using CPT+ is discussed and shown in example (See example 5.6) by training parts of the test SDB.

The following testing implementation with an empiric foundation gives further indications of the ESDA methods' applicability. However, no quantitative verification has been implemented for the data revelations, and the analysis is fully based on evaluator observation of SDB data and mining or predicting implementations.

For the test data, additional filtering was required after counting cmds found from the resulting SDB. The amount of *"modify_ button"*, tw and *"apply_ button"* occurences caused the formation of skewed data, shown in Fig 16., where 39% of database contents consisted of three cmds. Filtering the results yielded a more leveled action count shown in Fig 17. Additionally, this influences the average itemset and the distinct item per sequence found from the SDB, shown unfiltered in Table 18., and filtered in Table 19., as database statistics output.

Table 18: Test database statistics before filtering, using stripped representation of SPMF database statistics tool output from Fournier-Viger et al. (2014b)

========== SEQUENCE DATABASE STATS ==========
Number of sequences: 12667
SDB: Filtered: NA
Number of distinct items: 7695
Largest item id: 7869
Average number of itemsets per sequence: 14.8
Average number of distinct item per sequence: 6.9

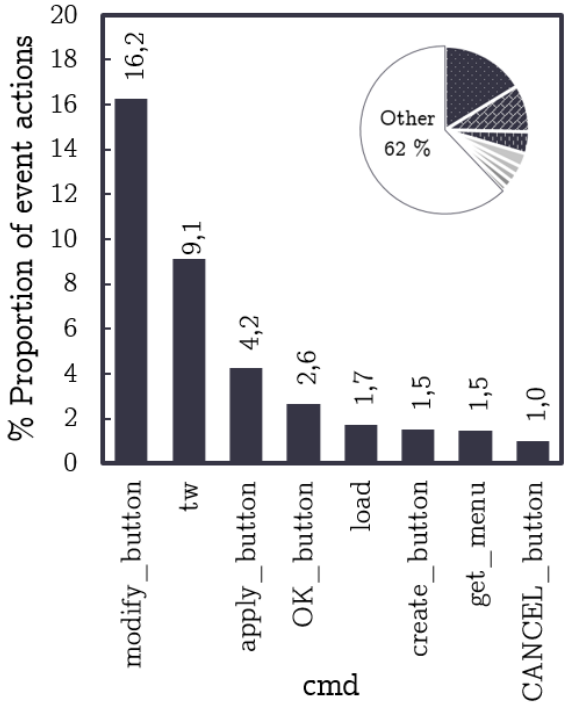


Figure 16: Proportion of test SDB highest occurring commandstarts before applying target filtering to limit overall percentage of known generic commands. *tw* button translates to dialog tab change and is filtered.

As seen in both Fig 17., particularly *"OK_ button"* remains relatively large with nearly 7% of the actions. The button will not be filtered as it serves as a sequence ending command by saving changed attributes to dialog memory and closing the detailing tool. The rest of the cmds are also considered as valuable information. The *"Load"* and *"get_ menu"* buttons are generic but often used as a prefix for loading predefined settings. Additionally, *"Create_ button"* and *"CANCEL_ button"* are often used as sequence ending cmds by either creating the desired detail or closing the detailing tools without applying any settings.

Table 19: Test database statistics after filtering *modify_ button, tw,* and *apply_ button cmds*, using stripped representation of SPMF database statistics tool output from Fournier-Viger et al. (2014b)

========== SEQUENCE DATABASE STATS ==========
Number of sequences: 12667
SDB : Filtered: modify_button, tw, apply_button
Number of distinct items: 7692
Largest item id: 7869
Average number of itemsets per sequence: 9.8
Average number of distinct item per sequence: 5.8

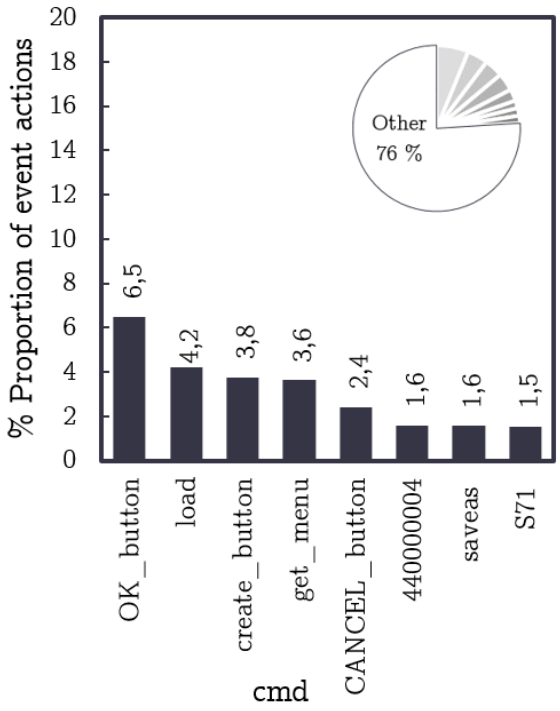


Figure 17: Proportion of test *SDB* commandstarts after applying target filtering. The number series represent a macro launched as a sub-process, and it is not identified as a detailing tool launch cmd.

5.1 Simple behavioral sequence model

Utilizing detectable sequence example outcomes for triggering events of interest to the evaluator was discussed in section 3.1. A simple detection model is shown in Fig 18. An example count done using the model is shown in Table 20. The *SDB* was mined for average repetition transition counts that could be used by the evaluator as a threshold of interest.

5.1.1 Case: Average transition of repetitive modify

As an example of the output of mining the *SDB* for modify patterns, the highest occurrence of modify event traversal was one traversal in 43% of all sequences. When limiting the scale to a 15% support level, repetitive modify occurred as a traversal pattern five times or less in a sequence. The tested example threshold creation was for *Repetitive modify*, finding cases such as in Table 7., captured from the *SDB*. In the mining result before filtering out *"modify_ button"*, the average repetitive modify was obtained from either single tools or in general and used as a problem indicator. The model example was a simple counting principle, as seen in Fig 18., discovering sequences that exceed the threshold values set by the evaluator.

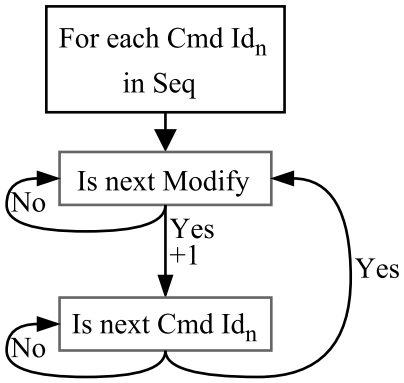


Figure 18: A simple model representation of counting repetitive modify action traversal between model 3D view, changing the same attribute. A count threshold triggering alert could be set by the evaluator using average repetitive modify counts found in *SDB*.

Table 20: Modify event traversal between model view taken from *SDB* sequence, (only relevant actions shown) changing the same attribute several times during single sequence capture in the detailing tool *Rectangular column reinforcement (83)*.

Pos	Timestamp	Cmd 1	Cmd 2	Occ
8	29.4.2016 16:49	akit.ValueChange	TopCornBarDist2	
13	29.4.2016 16:49	modify_button	modify_button	1
17	29.4.2016 16:50	akit.ValueChange	TopCornBarDist2	
18	29.4.2016 16:50	modify_button	modify_button	2
26	29.4.2016 16:52	akit.ValueChange	TopCornBarDist2	
27	29.4.2016 16:52	modify_button	modify_button	3
29	29.4.2016 16:52	akit.ValueChange	TopCornBarDist2	
30	29.4.2016 16:52	modify_button	modify_button	4
35	29.4.2016 16:52	akit.ValueChange	TopCornBarDist2	
36	29.4.2016 16:52	modify_button	modify_button	5
37	29.4.2016 16:53	akit.ValueChange	TopCornBarDist2	
38	29.4.2016 16:53	modify_button	modify_button	6

5.1.2 Conclusions: Problem identification

By utilizing the analysis methods and creating a threshold by observing sequential patterns found for the selected content, the pattern recognition using behavior models described seems possible. Using the threshold in repetitive patterns, such as minimum occurrence of five modify actions, there is a possibility to do inspections of all data or just parts of the *SDB*, such as the detailing tool of interest. The results would show all modify traversals over the given threshold. The data reveals sequential patterns and contains an identifier to see the full sequence contents from the database, similar to mining without a specific pattern.

5.2 Tool start and sequence relevance

Detailing tool command starts are the foundation of the sequence detection discussed in this paper. Sequence capture is different from recording a single event action, as it contains indexed actions in chronological order. Definitions of sequences and sequence starts were discussed mainly in Sections 3.2 and 4.3. Sequence examples are found in Fig 6., Fig 12. and in Table 17.

5.2.1 Case: Counting sequence starts

In order to evaluate the detailing tool information from the test SDB angle compared to the direct raw data counting of actions, a use case example was created to show individual start counts of most popular macro detailing tools. The test data included *Action starts* and *Sequence starts*, where both function as a command start for detailing tools. A total of 219529 action starts were added for comparison. The total number of sequences that contained a detailing tool sequence was taken from from test SDB, totalling 12668 individual sequences.

Table 21: Most frequent tool start count listing ordered by action starts.

Tool name	Action start	Sequence start
Unfold surface (21)	961	328
Border rebar for single edge (93)	943	423
Rectangular column reinforcement (83)	929	596
Embedded anchors (008)	909	677
Array of objects (29)	880	584
Pad footing reinforcement (77)	806	283
Beam reinforcement (63)	801	408
DWG profile to library (6)	788	727
Hole Generation (32)	746	653
Slab bars (18)	677	502
	8440	5181

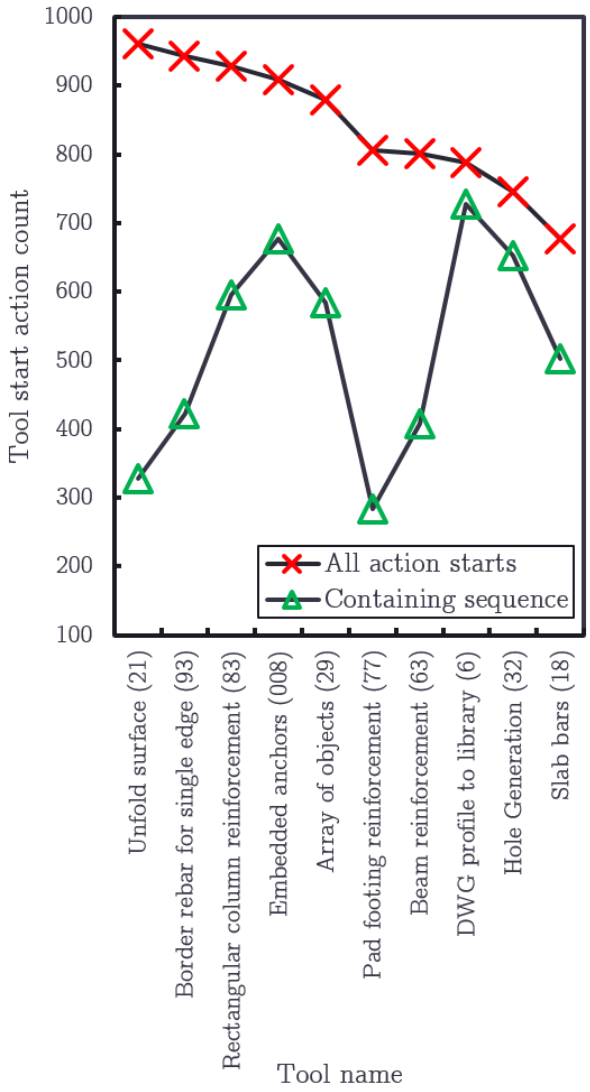


Figure 19: detailing tool start action counts. High action count does not correlate to actions counts that additionally contain tool UI-interaction.

5.2.2 Conclusions: detailing use case differences

The most frequently used detailing tools are shown in Table 21. and Fig 19. There is no correlation between the detailing tool starts that contain a sequence of UI-interaction and all found action starts. This might indicate that the tools are used for more generic tasks and do not require equal UI-interaction.

From these tools, *Embedded anchors (008)* and *DWG profile to library (6)* contain the highest amount of UI-interaction. By viewing the tool interfaces and sequence examples, both tools are frequently changed to suit the needs of each use case.

Additionally, sequential pattern mining of embedded anchors tool revealed[68] that the users utilize the existing pre-defined settings, but still require additional changes.

The tools *Unfold surface (21)* and *Pad footing reinforcement (77)* usage contains the lowest amount of sequences. The tool interfaces and basic use cases might indicate that the Unfold surface might function with generic one-time applied settings. The pad footing reinforcement tool has multiple dialog and setting options, but by viewing the structural design domain, pad footings are often similar by design.

From the sequence perspective, the results show highly different action start counts. Using SDB for statistical comparison, such as the most used detailing tools, might reveal valuable information for the evaluator. Additionally, users commonly test different detailing tools and their suitability for the workflow, possibly producing additional action counts.

5.3 Lower frequency workflow

The spectrum of the different frequency workflows as HCI events was previously discussed in Section 3.1 and shown in Fig 4. Frequency is also related to different levels of recognition depth, discussed in Section 4.2 and has played a high overall role when defining what action data is required to be captured by sequence detection.

5.3.1 Case: Chronologically ordered tool starts

A use case example of lower frequency workflow describing user detailing tool usage of several days was created using a single user log-file. Table 22. shows the workflow capture of eight working days. Found actions were calculated together, forming a count if no other tools were started in-between. The information captured was not limited to macro-tools only, additionally recording plugin and application start actions. Sequence detection was isolated as a separate test SDB. As there is a limitation in the mappings of plugin and application information, sequential pattern mining based analysis was not applicable.

[68] See 5.5.1, where mining was performed to reveal detailing paths

Table 22: Example capture from *SDB* containing chronologically ordered user detailing workflow information from eight consecutive working days.

Timestamp	Tool Name	Count
26.4.2016 7:08	WallLayoutConnector	5
-	WallLayoutElementation	2
-	WallLayoutOpening	4
-	WallLayoutSeam	4
-	WallLayout	2
-	FloorLayout	2
-	SlabReinforcementTool	2
-	WallLayout	2
-	SandwichWallWindow	4
-	SlabReinforcementTool	1
-	E Sheet Reference	3
-	Stair Drawing Views and Details	7
-	FloorTool	1
-	PEBMember	1
-	FloorLayoutDetailing	1
-	Similar Assemblies Filter	1
3.5.2016 17:30	SlopingSlabDrainage	7
8 working days	Total	49

5.3.2 Conclusions: Revealing common use workflows

The case example follows evaluator interest based predefined settings. Based on the interest, different frequencies can be analyzed, for instance ordered start commands without seeing the actual higher frequency sequence information. To fully utilize workflow analysis, the plugin and application sequences should be accessible. When enabled, sequences could be mined for sequential patterns on various levels. In the example case, a single user log data container can represent a low frequency action sequence. Different user based low frequency captures could be mined against each other in search of a common workflow. After a low level frequency lookup, the higher frequency information of the same data could be mined further for UI-interaction in detailing tool dialog structure. Similar approach also applies to general software UI actions.

5.4 Help contents used from detailing tools

Finding help after trying to use the detailing tool UI is considered as a disruption in both the workflow and the usability. Understanding why the help action is used connects to the pattern recognition discussed throughout this paper, as users have the tendency to try to solve issues using several attempts before frustration.

5.4.1 Case: Help actions calls inside sequence

Finding occurrences of help action instances was done by searching sequence contents for *"joint_help"* cmd. Found cmds were traced back to the original sequence start. The sequences were counted and listed shown in Table 23. Total number of tools inspected was the full testing SDB with the sequence count of 12668. A total of 178 help disruptions were discovered, of which the top ten tools represented 47, denoting to 26% of all searches for assistance in software.

Table 23: User action counts for "joint_help" to search assistance.

Tool Name	Count
Pilecap reinforcement (76)	6
Unfold surface (21)	6
DWG profile to library (6)	5
Concrete stairs (65)	5
160000079 (ext0)	5
Rectangular column reinforcement (83)	4
Triangles generation (19)	4
Generation of purlins (50)	4
90000064 (ext1)	4
3D cut (10)	4
Total	47

5.4.2 Conclusions: Cause of usability breakdown

Due to low counts of help usage, the results were re-inspected for a possible cause. It was discovered that the keyboard quick command "F1" that triggers help while in the detailing tool UI, is unmapped, and not present in raw event log. Instead, the user is required to press *"Help..."* button located tool dialog in the upper right corner. There was not enough data to perform sequential pattern analysis on the found sequences to reveal repetitive behavior in the UI structure before disruption. Shown in Table 23, even the top five tools have the average sequence count of less than six. There may be other reasons for low help usage. The detailing tool help is mostly[69] loaded from *Tekla User Assistance* (TUA) web service, also available for registered users via web browser. Users could be familiarized to search assistance directly from web pages. Macro tools also have a slightly smaller user base compared to modern plugins. Plugin help search behavior may be different but cannot be analyzed before mapping support for plugins.

Having a higher sequence count of each detailing tool help action could enable ESDA based analysis, revealing small patterns that lead the user seeking for help. These patterns could, for example, consist of certain parts of dialog attributes and modify traversal or follow other preliminary models described in Table 7. Identifying

[69]Some older detailing macros were found to contain local help page content

the exact cause enables simple dialog reconstructions or updated help contents for given tool parts causing disruption.

5.5 Revealing common detailing path

Software use behavior analysis using ESDA approaches is a primary topic in this paper. Mining UI paths as an example case is constructed upon discussed topics such as the selection of usability domain approaches in Section 2.2 and 3.2. The equivalent approaches as modern mining methods are introduced in Section 3.3 and the overall suitability of data and implementation discussed in Section 4. The example is thought to show scalable potential to other frequencies of events and parts of the software, where sequential pattern mining could be applied.

5.5.1 Case: Visualization of frequent tool (008) and (83) paths

The first detailing tool analyzed was *Embedded anchors (8)* shown in Fig. 20. The analysis process followed the model shown in Fig. 15. A filtered export was made from the testing *SDB* to contain only the sequences that were started using the detailing tool. The resulting analyzed sequence count was 677. Frequent sequence count was 29, varying in pattern lengths up to four, when using the support of 10%. Using the support of 5% frequent pattern length extended up to six, with 146 patterns. The highlighted area represents the most frequent commands found in the patterns. Typical use path involves both the blue and cyan path that occur in the *Placement* and *Input* dialogs. The analysis employed additional filtering of *"load"* and *"get_menu"* actions due to the high utilization of predefined settings.

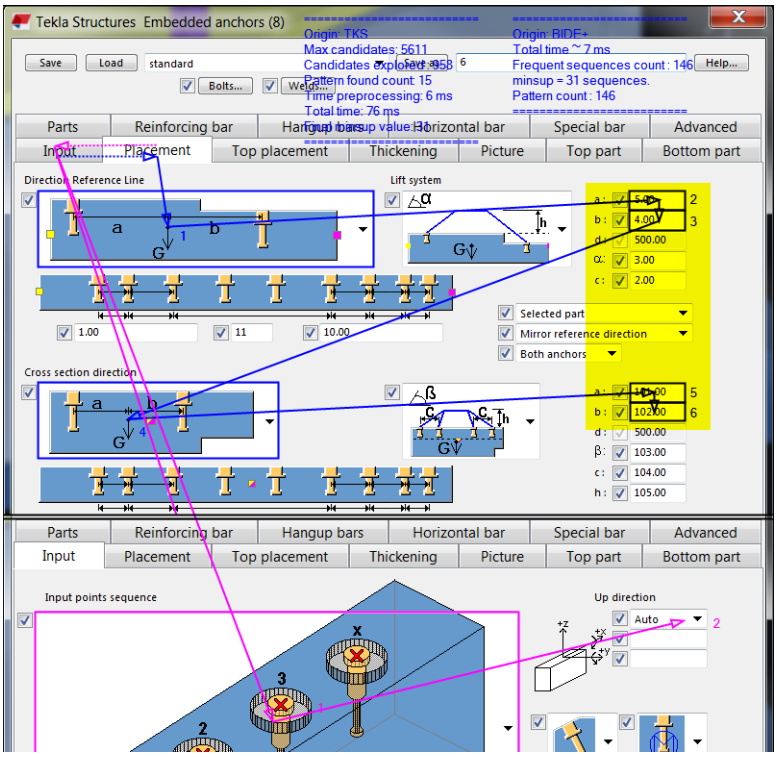


Figure 20: BIDE+ and TKS based results visualization of frequent user work path in *Embedded anchors (8)* tool. Path traversal is shown as numbers.

The second detailing tool analyzed was *Rectangular column reinforcement (83)* shown in Fig 21. Again, the analysis process followed the model shown in Fig. 15. A filtered export was made from testing the *SDB* to contain only the sequences that were started using the detailing tool. Resulting analyzed sequence count was 596. Frequent sequence count was 43, varying in pattern lengths up to three when using the support of 10%. Using support of 5%, frequent pattern length extended up to 6, with 244 patterns. The highlighted area represents the most frequent commands found in the patterns. The blue and green paths were the most popular ones, often continued by moving from the *Main bars* to the *Side bars* tab, the path marked here with cyan.

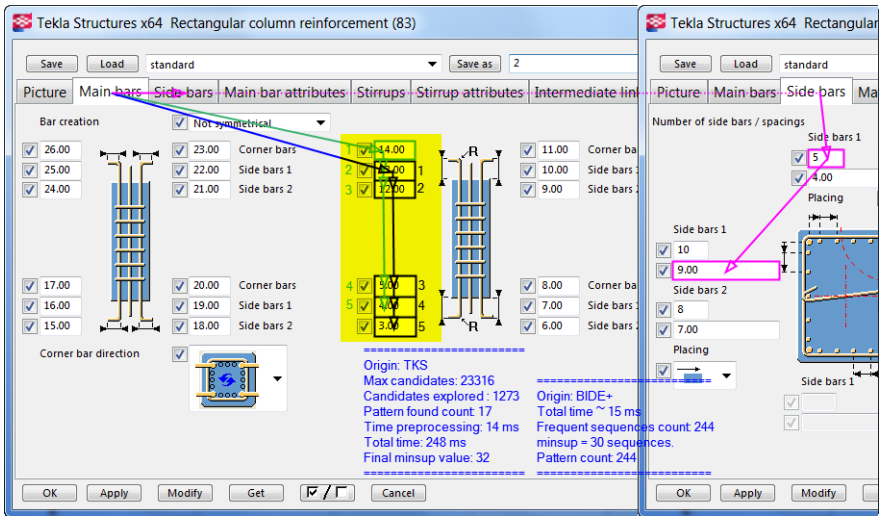


Figure 21: BIDE+ and TKS results visualization of frequent user work path in Rectangular column reinforcement (83) tool. Path traversal is shown as numbers.

5.5.2 Conclusions: Found software use behavior

The sequential pattern mining test implementation in this paper reveals software use behavior that concentrates on distinct detailing tools. The tested approach is also functional when searching the entire SDB for repetitive patterns that reveal the highest occurring patterns in all data, traceable back to source sequences. However, inspections to test SDB show that the majority of different detailing tools contain unique names for dialog content. As a result, different tools are not comparable for repetitive behavior.

Focusing on the elimination of the highly occurring irrelevant cmds after the first analysis iterations tends to output more valuable information of the detailing workflow. The additional filtering done to *Embedded anchors (8)* clarified the analysis. Filtering should be done in careful iterations as it helps understand the effect of each output sequential pattern information.

The sequence lengths and variance found after the mining showed some centralization of workflows. A single tool showed different usage patterns, making the overall repetitive pattern support lower and more fragmented. This might be due to the test data containing global usage sequences. The analysis could be more accurate if the SDB would be targeted to selected user groups, such as organizations or even parts of an organization to single user. Using this approach, the detailing task variation due to global differentiation could be minimized.

5.6 Prediction of detailing path

Predicting the user's next action in the detailing workflow was discussed as continuation for sequential pattern mining in Section 2.2.4. Selecting the approach and utilizing the prediction of next user actions in detailing workflow was discussed in Section 3.3.3. The example of prediction aims to show the scalable potential of utilizing sequence training based learning approaches for both interface related actions and background processes.

5.6.1 Case: Prediction visualization of tool (63)

Sequence prediction examples were tested for the detailing tool in three iterations. The analysis process followed the model shown in Fig 15., using a modified[70] CPT+ prediction approach with two different length *prefixes*. From testing SDB, a filtered export was made to contain only the sequences that were started by using the detailing tool. Resulting analyzed sequences count was 408. Additional filtering was made to reduce highly occuring single actions. Filtered *cmd "bar5_spacing"* is shown in Fig 22., labeled as *P3* and in Fig 23., labeled as *P1*. Number of distinct items was 914, item-sets per sequence 16, distinct items per sequence 9.3, and occurrences for each item 1.7. The resulting predictivity found was below four actions. Following predicted actions were looping duplicate single action suggestions caused by repetitive traversal stacking.

For the third iteration, a new filtering was utilized for the SDB by removing repetitive traversals in each sequence. As a result, no duplicate items appeared twice in the same sequence, arranged in first appearance order. Number of distinct items was 915, item-sets per sequence 9.4, distinct items per sequence 9.3. The resulting predictivity had no issues of single action looping.

[70] Source change that re-applied the prediction by user given length, using a previous prediction prefix and output prediction result

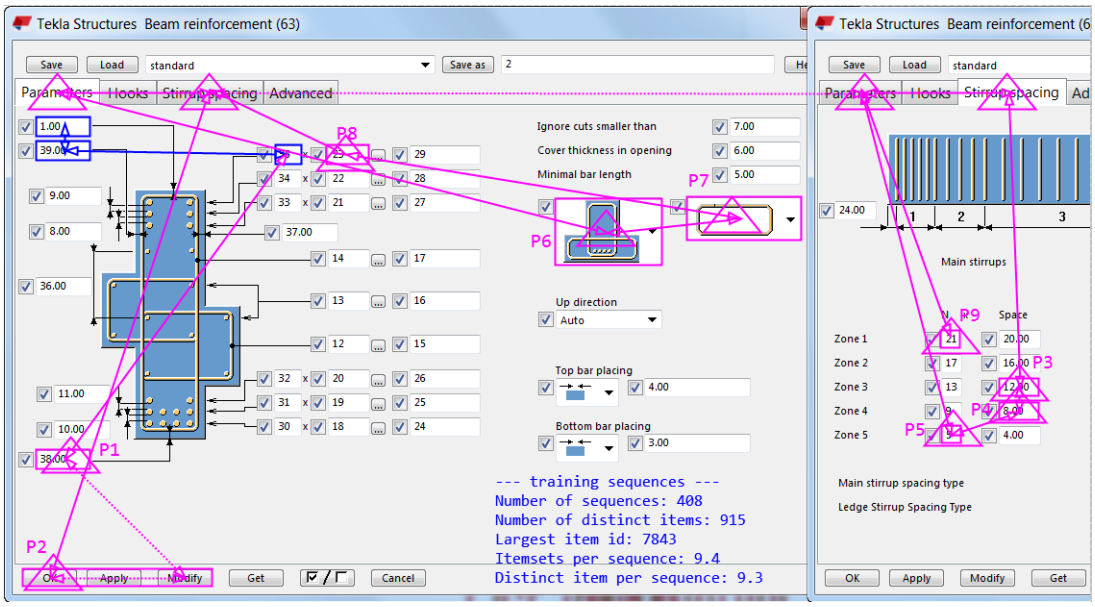


Figure 22: CPT+ sequence prediction of detailing tool *Beam reinforcement (63)*. Prediction outcome done using the prefix length of three.

First prefix length path in Fig 22., contains three actions and the second, shown in Fig 23., contains eight actions, boths shown in blue color. Corresponding prediction for the next items use cyan color marked with a symbol, prediction path P, followed by arrow path, square attribute marking and triangular marking representing exact prediction action. The prefixes mimic the user's detailing work, changing a load bearing concrete beam top and bottom reinforcing attributes. Cmds *"Apply_button"*, *"Modify_button"* and *"tw"* were filtered beforehand. Tab changes, and user "modify-apply-ok" routine were assumed and shown in dotted lines instead of actual prediction entries.

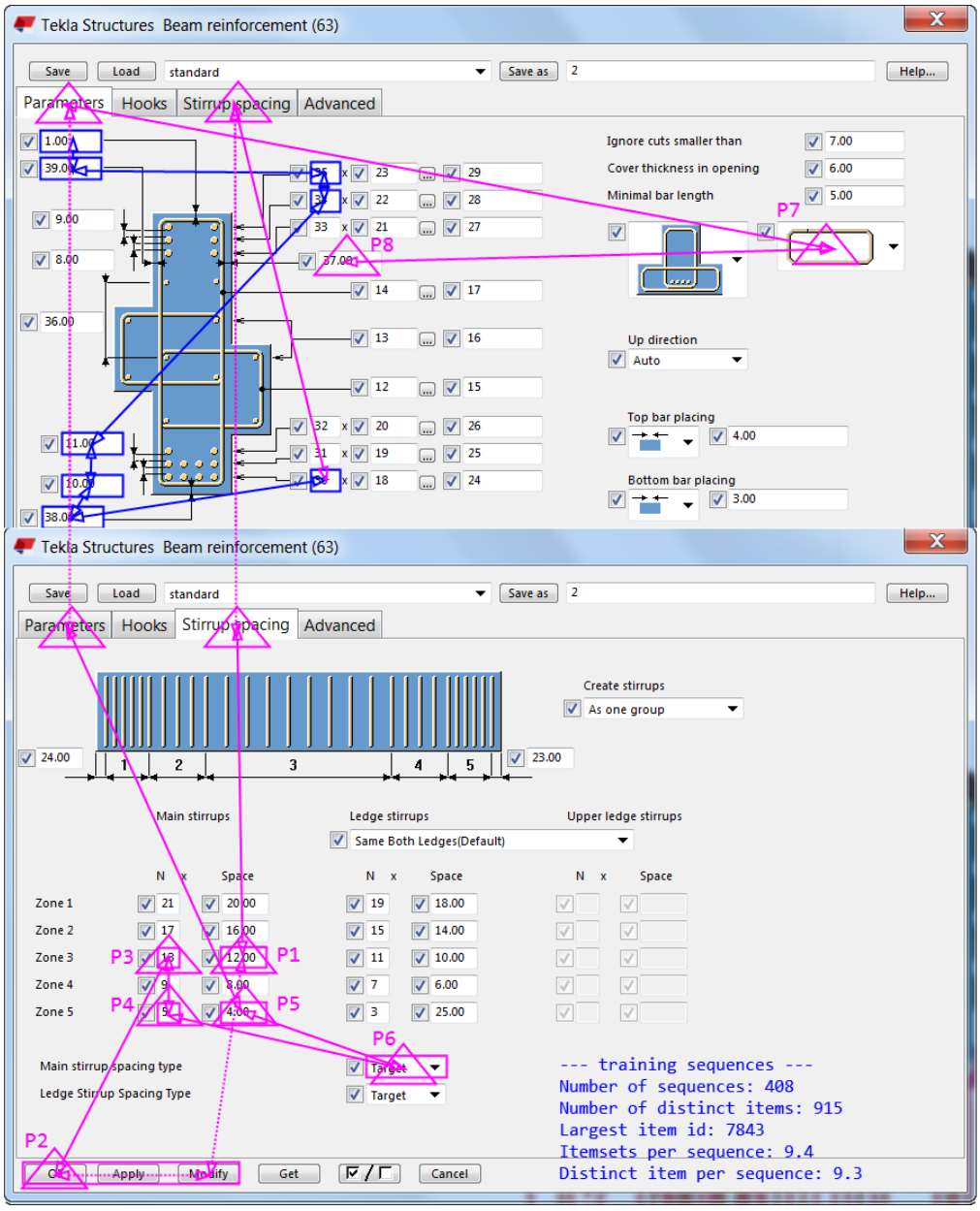


Figure 23: CPT+ sequence prediction of detailing tool *Beam reinforcement (63)*. Prediction outcome done using the prefix length of eight.

5.6.2 Conclusions: Prediction of workflow

The sequential patterns found in the data are trainable for pattern prediction. The CPT+ algorithm was able to predict test application data successfully. The prediction was also time efficient.[71] In the test case with the prefix length of three, the single next action duration using 408 sequences was 16 ms, with $2 + 10$ action

[71] Prediction was run both on Intel Core i7-4910MQ running Windows, and Intel Core i7-4770K running Linux. GPU's were not utilized for testing

total prediction time of 110 ms. In the test case with the prefix length of eight, the prediction duration for a single sequence was 15 ms, with $8 + 10$ action total prediction time of 4799 ms. This indicates that the prediction approach could be suitable for real-time application predicting.

Based on inspection, the prediction of the next attribute fields was in line with the prefix design task. Common actions in concrete beam detailing include reinforcing top and bottom section, followed by stirrups, especially middle spacing and typical high shear force areas after beam supporting points. Based on the inspection done using BIDE+, prediction using a short prefix seems to follow the most used overall commands since there is high variance in possible actions after a small prefix based search in the prediction tree. This was also noticeable in Fig 22., as a more unnatural workflow path, indicated by unnecessary tab changes. A longer prefix in Fig 23., seemed to predict real user workflow more accurately. As described in Section 3.3.3, found matches of the longer prefixes in the prediction tree decrease action variation and focus on predicting common behavior after a common prefix path.

To improve accuracy of the approach, further implementation is theorized. This require use of assigning model object information as prefix. In a sequence this would denote to prefix with itemset such as:

$$s_1 = \langle \{a, b, c\}, \{d\}, \{e\}, \{f\} \rangle$$

Instance of $\{a, b, c\}$ in this case is selected as any model object properties, for example *"Name","Profile"* or *"Material"*[72] and the rest as detaling tool interactions. The prediction would contain link between 3D-model objects subject to detaling and user detaling workflow initiated by the user. For CPT, this could further reduce amount of possible common sequence formation found in a prediction tree, focusing to previous sequences with similar object properties. A possible outcome could have impact on predicting correct structural detaling information based on object properties ultimately enabling accurate prediction of design values.

During testing, two challenges related to prediction accuracy and length were discovered. Similar to the previous cases using sequential patter mining, the tool *Beam reinforcement (63)* showed variance in workflows, resulting in fragmented sequence contents. The analysis could be more accurate if the SDB was targeted to selected user groups, such as organizations or even parts of an organization to single users. Additionally, filtered repetitive modify traversals only filter out single events from the SDB.[73] This results in many occurrences of actions stacking after one another. During prediction tree creation, the path traversal of the found prefix may contain stacked duplicates. Prediction continued to suggest the same actions. A solution was to filter out repetitive traversal caused duplicates in each sequence.

[72] See Test Application model objects definitions in Section 4.1

[73] See Table 20., where non-filtered modify traveral from the test data forms a high count of single cmds. A *"Modify_ button"* filtered event would cause the stacking of *"TopCorenBarDist2"* command

As a drawback, the filtered SDB loses information of repetitive behavior. Also, the filtering builds the sequence in first appearance order. Weighted position based ordering could allow for a more natural sequence build.

6 Discussion and conclusion

This section will summarize and examine the research work done, first by going through the research questions addressed in Section 1.3. After reviewing the questions, the validity and reliability of the study is discussed in retrospect. At the end of the section, future implementation requirements towards the test application and further research work are introduced.

6.1 Answers to research questions

The primary questions addressed in this thesis are answered as follows:

RQ1. For what are the found exploratory sequential data analysis (ESDA) usability methods best feasible in contemporary BIM applications?

Proposed ESDA methods collections in Ivory and Hearst (2001); Hilbert and Redmiles (2000); Cook and Wolf (1995); Sanderson and Fisher (1994); Cuomo (1994), previously discussed in Section 2.4, point out that the exploratory sequential data analysis results often are not anticipated but instead reveal how the software is used. Based on the inspection of the collection of available methods and revealed sequential test data[74] derived from the test application specific workflow sequence detection introduced in Section 4.4, there is a positive indication that the selected approaches Fisher's Cycles, LSA and MRP type ESDA, discussed in usability studies, and their comparative applied modern mining approaches BIDE+ and TKS[75] are feasible in contemporary BIM applications and the revealing of user workflow and repetitive software use behavior. The test *SDB* under observational analysis also seems to confirm the used working hypothesis introduced in Section 4.1, of general user workflow order while performing common detailing tasks.

RQ2. What kind of software usage information is revealed from sequential data?

Existing study in usability, introduced in Sections 2.2, and 2.3, propose that sequential data from repetitively occurring sequences of user actions show task information, which, when further analyzed, could reveal specific locations or action patterns causing causing usability related issues or disruptions in existing software systems. This background information was also used to create preliminary disruption outcome candidates for model creation, shown in Table 7., as information revealing examples. On the basis of the automated usability analysis method inspections[76], the selected sequential data utilizing approaches Fisher's Cycles, LSA and MRP, introduced in Section 3.2, all reveal occurrences of sequential patterns from collections of sequences that are chosen by the evaluator and could use different triggering for sequence start and ending. The existing studies propose that captured sequential patterns could reveal diverse frequent workflow information.

[74] See test data sequence examples in Fig. 6., Fig. 12. and in Table 17.
[75] See used mining approaches in Section 3.3
[76] See inspected methods in Section 2.4

Depending on the data filtering or model used in Secton 5., the proposed ESDA comparative mining approaches used against the captured detailing sequence SDB during test implementation are capable of revealing various kinds of information. For the test examples it was possible to 1) reveal model based usability disruption occurrences using sequential pattern mining, as shown in example 5.1, 2) utilize SDB statistically for gathering information about various sequence related counts and workflow information, as shown in examples 5.2, 5.3, and 5.4, and 3) utilize sequential pattern mining to reveal common detailing workflow, as shown in example 5.5.

RQ3. How can the obtained information be utilized in prediction?

Prediction that utilizes sequence data was introduced in Section 2.2.4. Implementable prediction model approach CPT+ was introduced in Section 3.3.3, predicting the next item of a found pattern prefix that is not introduced previously during any sequence training. With minor modification, the model was able to efficiently train and predict using available sequences in test SDB, as shown in example 5.6. Prediction done by utilizing matching given prefix and common sequential patterns found in SDB can minimize the variation of available possibilities and suggest more similar task related actions instead of counting overall action occurrences.

6.2 Validity and reliability

In terms of validity and reliability, the results of this paper can be divided into two distinct categories. In the first category, the inspections, analysis and selection of the existing automated usability analysis theorized approaches followed a qualitative path. Also, the results from sequential pattern mining were based on observation of commonly occurring patterns with high support. In the second category I have addressed that the amount of data[77] analyzed in this work is adequate when compared with commonly[78] used testing datasets in Fournier-Viger et al. (2014b). Also, to my knowledge there are no minimal functional thresholds for the used approaches.

In the first category, selection of suitable approaches was guided by existing studies and preliminary requirements for the test application. The coverage of approaches in the usability paradigm was 23 different approaches, listed in Table 6. It was found that there was a relatively small amount of information found after Ivory and Hearst's collection[79] of automated evaluation techniques. In retrospective, newer studies for modern software system automated usability analysis would have provided better comparative reasoning throughout the paper. Quite the contrary, from a pure data analysis perspective, the available approaches possibly adept for further investigation were substantial. In this case, the depth required to understand different algorithmic approaches, comparative to found usability studies ESDA approaches, and the implementation for testing resulted in time strain, reducing the

[77] See Section 4.3, Available Data
[78] Datasets tested for BIDE+, TKS and CPT+ in Fournier-Viger et al. (2014b)
[79] See collection in Ivory and Hearst (2001)

possibility for more extensive searches.

In the second category, using authentic work based workflow data was used as the basis for all results. Sequence capture was manually validated against raw event logs, and after iterative adjustments, both the detection algorithm and the used recording scope[80] led to the situation where there was no loss of detailing tool related data. As the recording scope was broader, the missing plugin and applications usage information reduced the initial sequence size. The correct mappings and using the initial 26.3 GB of raw user event data would have significantly increased the sequence yield.

The last data-related noticeable element regarding this study was the quality of the obtained data. The extent to which the raw event data is from testing users or doing something other than actual detailing work, such as developing new tools or processes, can only be argued. However, the issues of this data should decrease when increasing the data-set size, as overall, the users of the software mostly produce real user worfklow data, resulting from the natural usage of the inspected tools. Also, as discussed previously in the results of the common detailing path in 5.5, and the prediction of the detailing path 5.6, the currently obtained event data is global, causing some increased variance in common detailing use behaviour.

6.3 Future work

The results and implementations done during this thesis should offer a range of scalable use cases and subjects for future work. During the course of the study, functional ESDA usability analysis approaches suitable for contemporary BIM applications were found. The approaches were also tested and provided information of how the software is used. Additionally, the revealed information has opened possibilities for predicting user actions in software.

As minimal requirements I will first propose future work that is targeted for the test application. Utilizing this paper's results, simple graphical application to produce evaluator desired basic filtering, filtering models creation and visualization should be constructed. As described previously in Section 2.2, *Deployment* or the production of *"quantitative or qualitative summaries"* is a fundamental part of any analysis process. At current state, the implementations are run in a source version that is difficult to use, and the visualizations are evaluator created, based on translated event action results of sequential data mining. Overall, the example visualizations provided in Section 5., could work as templates for development.

A next step for test application after creating a entry point for usability evaluators, is to map all missing actions from the recorded raw-data, where most important are *Plugins*, *Applications*, all their corresponded attribute values[81] content

[80] See the scope of data both in Section 1.2, and 4.3

[81] In this context, user applied attribute values in *textboxes, comboboxes* and test application specific *catalog-selection* boxes

from users and newly created WPF-commands. This would open up wide-ranging analysis possibilities such as discussed in 5.3, as lower frequency workflow capture. Additionally, the test application drawing and documentation module should also be considered to be mapped. Utilizing all available user-actions could enable full analysis software interface and interactions.

As future research work related to obtained data, I propose further inspections towards statistical analysis and verification. Further validation could be done towards the quality and possible use cases of statistical data obtained directly from SDB or the resulting sequential pattern mining targeted to BIM workflow. Also, work towards validating prediction related accuracy by incorporating common ethnographic usability testing methods is considered an interesting topic. This might also lead to the discovery of new use cases for sequential analysis based prediction.

For a direct continuum in method implementations and their further proceedings, I propose additional development, particularly towards the prediction of user workflow. The current implementation was able to predict and mimic user workflow as shown in example 5.6. As a next step, it would be beneficial to investigate how to incorporate model side detailing tool object selections as prefixes to enable structural object based categorization as a part of prediction. Obtaining additional prefix information from what the user is currently interacting with, could open several applications for sequential pattern prediction. Applications naturally include detailing workflow, but additionally, highly repetitive work in test application exists when creating structural documentations such as drawings.

References

Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA. IEEE Computer Society.

Akers, D., Jeffries, R., Simpson, M., and Winograd, T. (2012). Backtracking events as indicators of usability problems in creation-oriented applications. *ACM Trans. Comput.-Hum. Interact.*, 19(2):16:1–16:40.

Alpaydin, E. (2014). *Introduction to Machine Learning*. The MIT Press, 3nd edition.

Behrens, J. T. (1997). Principles and procedures of exploratory data analysis. *Psychological Methods*, 2(2):131 – 160.

Bramer, M. (2007). *Principles of Data Mining (Undergraduate Topics in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Brooke, J. (2013). Sus: A retrospective. *J. Usability Studies*, 8(2):29–40.

Cook, J. E. and Wolf, A. L. (1995). Automating process discovery through event-data analysis. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 73–82, New York, NY, USA. ACM.

Cuesta, H. (2013). chapter Practical Data Analysis.

Cuomo, D. L. (1994). Understanding the applicability of sequential data analysis techniques for analysing usability data. *Behaviour & Information Technology*, 13(1-2):171–182.

Dix, A., Finley, J., Abowd, G., and Beale, R. (1998). *Human-computer Interaction (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Eastman, C., Teicholz, P., Sacks, R., and Liston, K. (2008). *BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors*. Wiley Publishing.

Ehrlich, K. and Rohn, J. A. (1994). Cost-justifying usability. chapter Cost Justification of Usability Engineering: A Vendors's Perspective, pages 73–110. Academic Press, Inc., Orlando, FL, USA.

Ericsson, K. A. and Simon, H. A. (1993). *Protocol analysis verbal reports as data*.

Fisher, C. (1991). Protocol Analyst"s Workbench: Design and evaluation of computer-aided protocol analysis.

Fisher, C. and Sanderson, P. (1993). Exploratory sequential data analysis: Traditions, techniques and tools. *SIGCHI Bull.*, 25(1):34–40.

Fournier-Viger, P., Chun-Wei Lin, J., Kiran, R. U., Koh, Y. S., and Thomas, R. (2017). A survey of sequential pattern mining. *Ubiquitous International*, 1(1):54–76.

Fournier-Viger, P., Gomariz, A., Campos, M., and Thomas, R. (2014a). *Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information*, pages 40–52. Springer International Publishing, Cham.

Fournier-Viger, P., Gomariz, A., Gueniche, T., Mwamikazi, E., and Thomas, R. (2013). *TKS: Efficient Mining of Top-K Sequential Patterns*, pages 109–120. Springer Berlin Heidelberg, Berlin, Heidelberg.

Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu., C., and Tseng, V. S. (2014b). SPMF: a Java Open-Source Pattern Mining Library. *Journal of Machine Learning Research (JMLR)*, 15:3389–3393.

Gould, J. D. and Lewis, C. (1985). Designing for usability: Key principles and what designers think. *Commun. ACM*, 28(3):300–311.

Grudin, J. (1991). Systematic sources of suboptimal interface design in large product development organizations. *Hum.-Comput. Interact.*, 6(2):147–196.

Guan, Z., Lee, S., Cuddihy, E., and Ramey, J. (2006). The validity of the stimulated retrospective think-aloud method as measured by eye tracking. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 1253–1262, New York, NY, USA. ACM.

Gueniche, T., Fournier-Viger, P., Raman, R., and Tseng, V. S. (2015). Cpt+: Decreasing the time/space complexity of the compact prediction tree. In *Proc. 19th Pacific-Asia Conf. Knowledge Discovery and Data Mining*, pages 625–636. Springer.

Gueniche, T., Fournier-Viger, P., and Tseng, V. S. (2013). Compact prediction tree: A lossless model for accurate sequence prediction. In *Proc. 9th Intern. Conference on Advanced Data Mining and Applications*, volume 2, pages p.177–188. Springer.

Han, J. (2012). *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Hartwig, F. (1979). chapter Exploratory Data Analysis.

Helfrich, B. and Landay, J. A. (1999). Quip: Quantitative user interface profiling.

Hilbert, D. M. and Redmiles, D. F. (1998). An approach to large-scale collection of application usage data over the internet. In *Proceedings of the 20th International Conference on Software Engineering*, ICSE '98, pages 136–145.

Hilbert, D. M. and Redmiles, D. F. (2000). Extracting usability information from user interface events. *ACM Comput. Surv.*, 32(4):384–421.

ISO-20282-2:2013 (2013). SO/TS 20282-2:2013 Usability of consumer products and products for public use. *ISO Standard*, Part 2.

Ivory, M. Y. and Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4):470–516.

Karat, C.-M. (1994). Cost-justifying usability. chapter A Business Case Approach to Usability Cost Justification, pages 103–140. Academic Press, Inc., Orlando, FL, USA.

Lecerof, A. and Paterno, F. (1998). Automatic support for usability evaluation. *IEEE Transactions on Software Engineering*, 24(10):863–888.

Macleod, M. and Rengger, R. (1993). The development of drum: A software tool for video-assisted usability evaluation. In *In Proceedings of HCI'93*, pages 293–309. Cambridge University Press.

Mansouri-Samani, M. and Sloman, M. (1997). GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering*, 4(2):96–108.

Mayhew, D. (1999). *The Usability Engineering Lifecycle: A Practitioner's Handbook for User Interface Design.* Interactive Technologies Series. Morgan Kaufmann Publishers.

Mayhew, D. J. and Tremaine, M. (1994). Cost-justifying usability. chapter A Basic Framework, pages 41–101. Academic Press, Inc., Orlando, FL, USA.

Mooney, C. H. and Roddick, J. F. (2013). Sequential pattern mining – approaches and algorithms. *ACM Comput. Surv.*, 45(2):19:1–19:39.

Muller, M. J., Dayton, T., and Root, R. (1993). Comparing studies that compare usability assessment methods: An unsuccessful search for stable criteria. In *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, CHI '93, pages 185–186, New York, NY, USA. ACM.

Myatt, G. J. and Johnson, W. P. (2014). *Making Sense of Data I.* ProQuest Ebook Central.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 373–380, New York, NY, USA. ACM.

Nielsen, J. (1993). *Usability engineering.* Morgan Kaufmann, USA.

Nielsen, J. and Landauer, T. K. (1993). A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 206–213, New York, NY, USA. ACM.

Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 249–256, New York, NY, USA. ACM.

Olson, G. M., Herbsleb, J. D., and Rueter, H. H. (1994). Characterizing the sequential structure of interactive behaviors through statistical and grammatical techniques. *Hum.-Comput. Interact.*, 9(4):427–472.

Pitkow, J. and Pirolli, P. (1999). Mininglongestrepeatin g subsequencestopredict worldwidewebsurfing. In *Proc. USENIX Symp. On Internet Technologies and Systems*, page 1.

Rajanen, M. and Iivari, N. (2007). *Human-Computer Interaction – INTERACT 2007: 11th IFIP TC 13 International Conference, Rio de Janeiro, Brazil, September 10-14, 2007, Proceedings, Part II*, chapter Usability Cost-Benefit Analysis: How Usability Became a Curse Word?, pages 511–524. Springer Berlin Heidelberg, Berlin, Heidelberg.

Rasmussen, J., Kim, and Vicente, J. (1989). Coping with human errors through system design: Implications for ecological interface design. *International Journal of Man-Machine Studies*, pages 517–534.

Rauterberg, M. (1993). Amme: an automatic mental model evaluation to analyse user behaviour traced in a finite, discrete state space. *Ergonomics*, 36(11):1369–1380. PMID: 8262030.

Rieman, J., Franzke, M., and Redmiles, D. (1995). Usability evaluation with the cognitive walkthrough. In *Conference Companion on Human Factors in Computing Systems*, CHI '95, pages 387–388, New York, NY, USA. ACM.

Rosenbaum, S. (1989). Usability evaluations versus usability testing: when and why? *IEEE Transactions on Professional Communication*, 32(4):210–216.

Rosenblum, D. S. (1991). Specifying concurrent systems with tsl. *IEEE Software*, 8(3):52–61.

Rubin, J. (1994). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition.

Sackett, G., of Child Health, N. I., (U.S.), H. D., of Washington. Child Development, U., and Center, M. R. (1978). *Observing Behavior: Data collection and analysis methods*. Observing Behavior: Proceedings of the Conference, Application of Observational/ethological Methods to the Study of Mental Retardation, Held at Lake Wilderness, Washington in June, 1976. University Park Press.

Sammut, C. and Webb, G. I., editors (2010). *Sequential Data*, pages 902–902. Springer US, Boston, MA.

Sanderson, P. M. and Fisher, C. (1994). Exploratory sequential data analysis: Foundations. *Hum.-Comput. Interact.*, 9(4):251–317.

Scarpa, B. and Azzalini, A. (2012). *Data Analysis and Data Mining*. Oxford University Press, USA. Oxford University Press, USA.

Schmettow, M. (2008). Heterogeneity in the usability evaluation process. In *Proceedings of the 22Nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction - Volume 1*, BCS-HCI '08, pages 89–98, Swinton, UK, UK. British Computer Society.

Schmettow, M. (2012). Sample size in usability studies. *Commun. ACM*, 55(4):64–70.

Siochi, A. (1989). *Computer-based User Interface Evaluation by Analysis of Repeating Usage Patterns in Transcripts of User Sessions*. Virginia Polytechnic Institute and State University.

Siochi, A. C. and Ehrich, R. W. (1991). Computer analysis of user interfaces based on repetition in transcripts of user sessions. *ACM Trans. Inf. Syst.*, 9(4):309–335.

Siochi, A. C. and Hix, D. (1991). A study of computer-supported user interface evaluation using maximal repeating pattern analysis. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 301–305, New York, NY, USA. ACM.

Sun, R. and Giles, L. C. (2001). *Sequence Learning : Paradigms, Algorithms, and Applications*, volume 1828 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg.

Tukey, J. W. (1977). *Exploratory Data Analysis*. Addison-Wesley.

Tzvetkov, P., Yan, X., and Han, J. (2005). TSP: mining top-k closed sequential patterns. *Knowl. Inf. Syst.*, 7(4):438–457.

Wang, J. and Han, J. (2004). Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering*, ICDE '04, pages 79–, Washington, DC, USA. IEEE Computer Society.

Wang, J., Han, J., Lu, Y., and Tzvetkov, P. (2005). Tfp: an efficient algorithm for mining top-k frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 17(5):652–663.

Zettlemoyer, L. S., St. Amant, R., and Dulberg, M. S. (1999). Ibots: Agent control through the user interface. In *Proceedings of the 4th International Conference on Intelligent User Interfaces*, IUI '99, pages 31–37, New York, NY, USA. ACM.