

Hugo Fooy

SOFTWARE USAGE DATA VISUALIZATION

Faculty of Computer Science
Master of Science Thesis
June 2019

ABSTRACT

Hugo Fooy: Software Usage Data Visualization
Master of Science Thesis
Tampere University
Software Engineering
June 2019

This thesis aims at investigating the adequacy of the Unified Model for Software Engineering Data and its technical framework for developing visualizations of software usage data. Two visual notations were developed using the aforementioned framework and its visualization templates. The data source was provided by logs of the software Kactus2 that had been previously collected. The two visualizations were evaluated both on a semantic level with an ontological analysis (based on the BWW-model), and on a syntactic level with the Physics of Notations. They were also presented to developers of Kactus2 for an additional assessment of their usability and usefulness. The results indicate that the data model and framework are indeed adequate for visualizing complex usage data from Kactus2. Furthermore, the visualizations appear to be both easy to understand and useful (in the sense that they provide insight to the usage of the software) by the developers. Based on these results, we argue that software visualizations of usage data in general - and in particular using the Unified Model for Software Engineering Data - should be studied and developed further as they may help improve software engineering products and processes.

Keywords: software visualization, visual notation, usage data, post-deployment data, physics of notations, ontological analysis, syntactic analysis

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

First and foremost, I would like to express my most profound gratitude to Prof. Kari Systä for the attentive supervision of this thesis. Thank you for giving me the occasion to work on this topic, and for encouraging me throughout the year.

I would also like to kindly thank Anna-Liisa Mattlia and Esko Pekkarinen, for answering my many questions and spending some of their precious time for me.

Many thanks to my friend and colleague Sami Ben Mariem for his proof-reading of this thesis, acute critics, and unconditional friendship.

Thank you also to the developers of Kactus2 who accepted to answer my survey.

Thanks to my friends and family for their support throughout the year, and especially to my tutor Timo Stolt.

Finally, a huge thank you to the ESN INTO club. Both for being like a family to me this whole year in Finland. Both for the club room which saw me writing most of this thesis, and the relaxing games that followed each writing session. Thank you Kitty for watching over me while all this happened.

Tampere, 14th June 2019

Hugo Fooy

CONTENTS

List of Figures	vi
List of Tables	viii
List of Symbols and Abbreviations	x
1 Introduction	1
1.1 Defining Goals	2
1.2 Research Questions	2
2 Theoretical Background	3
2.1 Software Analytics	3
2.1.1 Software and Analytics	3
2.1.2 Types of Software Analytics	4
2.1.3 Needs and Applications of Software Analytics	5
2.2 Post-Deployment Data	6
2.2.1 Software Data	6
2.2.2 Applications of PDD	7
2.3 Data Collection	8
2.3.1 Challenges and Limitations of PDD Collection	8
2.3.2 Technological Approaches	9
2.3.3 Selecting the Right Approach	10
2.4 Data Visualization	10
2.4.1 Designing Visual Notations	10
2.4.2 Designing Information Models	19
2.4.3 Semantic Analysis of Visual Notations	20
2.4.4 Software Visualization Today	21
2.4.5 Usage Data Visualization	22
2.5 Data Modeling	24
3 Research Material	26
3.1 Data Source	26
3.1.1 Kactus2	26
3.1.2 Data Collection	27
3.1.3 Analytics	28
3.2 Technical Framework	31
3.2.1 Database	31
3.2.2 Visualization Plugin	31
3.2.3 Applications of the Framework and Visualization Plugin	32
4 Methodology	34
4.1 Data Model	34

4.1.1	Elements of the Data Model	34
4.1.2	Links Between the Elements of the Data Model	36
4.1.3	Data Parsing and Linking	36
4.2	Visualizations	37
4.2.1	Session Timelines	37
4.2.2	User Timeframe	38
4.3	Evaluation	39
4.3.1	Semantic Analysis	40
4.3.2	Syntactic Analysis	41
4.3.3	Questionnaire	41
5	Results	44
5.1	Implementation	44
5.1.1	Technical Framework	44
5.1.2	Visualizations	45
5.2	Ontological Analysis	46
5.2.1	Redundant Constructs	47
5.2.2	Construct Deficit	48
5.2.3	Construct Overload	49
5.2.4	Construct Excess	49
5.3	Syntactic Analysis	49
5.3.1	Semiotic Clarity	50
5.3.2	Perceptual Discriminability	52
5.3.3	Semantic Transparency	54
5.3.4	Complexity Management	54
5.3.5	Cognitive Integration	55
5.3.6	Visual Expressiveness	56
5.3.7	Dual Coding	56
5.3.8	Graphic Economy	57
5.3.9	Cognitive Fit	57
5.3.10	Interactions Between the Principles	57
5.4	Questionnaire	58
6	Discussion	60
6.1	Interpretation of the Results	60
6.1.1	Ontological Analysis	60
6.1.2	Syntactic Analysis	61
6.1.3	Questionnaire	63
6.2	Research Questions	63
6.3	Limitations	64
6.4	Future work	65
7	Conclusion	66
	References	67
	Appendix A Actions recorded in the logs	72

Appendix B	BWW ontological concepts	73
Appendix C	Visualization semantic constructs	77
Appendix D	Representation mapping	79
Appendix E	Interpretation mapping	82

LIST OF FIGURES

2.1	Overview of analytics, adapted from [24].	4
2.2	Classification of analytical questions, from [12].	5
2.3	The refined collection framework for user-interaction collecting techniques, from [54].	10
2.4	Ontological mapping, from [41]. There should be a 1:1 mapping between the concepts and the constructs to avoid construct deficit, redundancy, overload, and excess.	12
2.5	Descriptive theory, from [41].	12
2.6	Prescriptive theory principles, from [41]. The honeycomb structure illustrates the desire of the authors to encourage modifications and extensions to the principles by further research.	13
2.7	Semiotic (non-)clarity: (a) symbol redundancy, (b) symbol overload, (c) symbol deficit, (d) symbol excess.	14
2.8	Cognitive integration: multiple diagrams are used to represent a system. . .	16
2.9	The 8 visual variables, from [4].	17
2.10	Hybrid symbols: implementing dual coding by using a combination of text and graphics.	17
2.11	Interactions between the principles, from [41]. + indicates a positive effect, - a negative effect, and +/- means the effect is either positive or negative depending on the situation.	19
2.12	Ontological completeness (left) and ontological incompleteness or construct deficit (right).	21
2.13	Ontological completeness with from left to right: (a) construct overload, (b) construct redundancy, (c) construct excess.	21
2.14	Mash-up software visualization, from [35].	23
2.15	Feature Usage Diagram, from [32].	24
2.16	UI heat-map, from [33].	24
2.17	Graphical representation of the Unified Model for Software Engineering Data, from [36] (updated).	25
3.1	An example of the Kactus2 Interface.	27
3.2	Event path graph.	29
3.3	Time for <i>document opened</i> to <i>unlocked</i> , per session (in seconds).	30
3.4	Implementation of the framework, from [36] (updated). Issue Collectors and Linkers are examples of usages of this framework.	31
3.5	Visualization of test log information, from [36].	32
3.6	Visualization of usage logs, from [36].	32

3.7	Visualization of issue management data, from [21].	33
4.1	Implementation of the Unified Data Model for Software Engineering Data. Yellow elements are related to <i>documents</i> , orange to <i>help pages</i> , red to <i>sessions</i> . Purple are <i>features</i> , blue are <i>users</i> , green is the <i>origin</i>	35
4.2	Implementation of the integration of the data in the database.	37
4.3	Session Timelines visualization.	38
4.4	User Timeframe visualization.	39
5.1	Source code structure: data sources. In green are the new folders and files created. In blue is the data source.	45
5.2	Starting UI of the User Timeline visualization.	46
5.3	Source code structure: frontend. In green are the new files created for the User Timelines visualization. In purple are the new files created for the Session Timeframes visualization. In orange are the existing files modified. Three dots indicate files of other visualizations that we are thus not using.	47
5.4	Visual variables used in the visual notations. Texture and orientation are not used. (*)The brightness is only used in the User Timeline visualization.	53
5.5	Semantically translucent notation: lines (constructs) and circles (events) placed along a timed-axis.	54
5.6	Example of modularization with the time-selector in the User Timeline vi- sualization. On the left: the default visualization. On the right: the same visualization zoomed-in between 12:35 and 12:55.	55

LIST OF TABLES

2.1	Summary of the techniques evaluations, from [54].	11
2.2	Studied aspects in software visualization research, from [34].	22
3.1	Session times between two actions (in seconds).	29
3.2	Statistics on toolbar button presses.	30
4.1	Questions on the Session Timeframe visualization. Questions were asked in this exact order.	41
4.2	Questions on the User Timeline visualization. Questions were asked in this exact order.	42
5.1	Technically redundant semantic constructs.	48
5.2	Visual vocabulary of the <i>Session Timeframe</i> visualization.	50
5.3	Visual grammar of the <i>Session Timeframe</i> visualization.	50
5.4	Visual vocabulary of the <i>User Timeline</i> visualization.	51
5.5	Visual grammar of the <i>User Timeline</i> visualization.	51
5.6	Answers to questions on the Session Timeframe visualization.	58
5.7	Answers to questions on the User Timeline visualization.	59
B.1	Basic concepts in the BWW-model: Things and properties.	73
B.2	Basic concepts in the BWW-model: Natural and human laws.	73
B.3	Basic concepts in the BWW-model: Classes and natural kinds.	74
B.4	Basic concepts in the BWW-model: States, events and transformations.	74
B.5	Basic concepts in the BWW-model: State and transformation laws.	74
B.6	Basic concepts in the BWW-model: Internal/external events and stable/un- stable states.	75
B.7	Basic concepts in the BWW-model: State and event spaces.	75
B.8	Basic concepts in the BWW-model: Coupling.	75
B.9	Basic concepts in the BWW-model: Composites and systems.	76
C.1	Basic concepts in the semantic-model: Entities.	77
C.2	Basic concepts in the semantic-model: Actions.	77
C.3	Basic concepts in the semantic-model: Objects.	78
C.4	Basic concepts in the semantic-model: Relations.	78
D.1	Representation mapping of the BWW-model: Things and properties.	79
D.2	Representation mapping of the BWW-model: Natural and human laws.	79
D.3	Representation mapping of the BWW-model: Classes and natural kinds.	80

D.4	Representation mapping of the BWW-model: States, events and transformations.	80
D.5	Representation mapping of the BWW-model: State and transformation laws.	80
D.6	Representation mapping of the BWW-model: Internal/external events and stable/unstable states.	80
D.7	Representation mapping of the BWW-model: State and event spaces. . . .	81
D.8	Representation mapping of the BWW-model: Coupling.	81
D.9	Representation mapping of the BWW-model: Composites and systems. . .	81
E.1	Interpretation mapping of the semantic-model: Entities.	82
E.2	Interpretation mapping of the semantic-model: Actions.	82
E.3	Interpretation mapping of the semantic-model: Objects.	83
E.4	Interpretation mapping of the semantic-model: Relations.	83

LIST OF SYMBOLS AND ABBREVIATIONS

AOP	Aspect-Oriented Programming
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
BWW	Bunge-Wand-Weber model
EDA	Electronic Design Automation
FPGA	Field-Programmable Gate Array
GDPR	General Data Protection Regulation
GoM	Guidelines of Modeling
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
IS	Information System
MCAPI	Multicore Communications API
MP	Microprocessor
MP-SoC	Multiprocessor System on a Chip
PDD	Post-Deployment Data
PoN	Physics of Notations
REST	Representational State Transfer
SE	Software Engineering
SME	Small and Medium-size Enterprises
SoC	System on a Chip
SOK	Software Operation Knowledge
UI	User Interface
UMSED	Unified Model for Software Engineering Data
XML	Extensible Markup Language

1 INTRODUCTION

Software visualization is a branch of Software Analytics (SA), which is the use of software data to enable better decision-making. Analytics allow managers and engineers to rely on rational information rather than intuition. Through automated tools, the large amount of data that originates from software can be turned into insight, improving the software development processes and products. As the field of SA develops, software analytics have been classified by various authors [2, 12, 39]: how and why they are performed, who they are performed for, or the analytical questions driving the process are several ways to categorize SA. Different types of software analytics call for different metrics. The choice of the metrics is related to the usage that will be made of the software data [10].

Software engineering produces a huge amount of data, from its early planning phase through its development, and while being used [9]. In this thesis, we will focus on Post-Deployment Data (PDD) or *usage data*. This is data that results from the interaction of the user with the software, after its deployment. Such data may come from logs, bug reports, stack traces, etc. To this date, PDD has mainly been used in automatic applications testing, or to analyse user behaviour. Feature usage is also a very well know application of PDD analytics.

Data collection is a non-trivial preliminary step to all analytics. How software data collection can be implemented is a vast subject, thus we will only focus on the collection of PDD. The collection of usage data comes with intrinsic challenges and limitations, that must be addressed. Hence, the specifics of the target and context for the application should be carefully considered when selecting a data collection technique [56]. Suonsyrjä presented a selection framework for automated usage data collection technologies [54].

Visualization is crucial to software analytics for helping people process the data in a more efficient way. This is especially true when information has to be shared between people with different levels of expertise, for example developers and stakeholders. Designing good visualizations is therefore a major issue and should be taken seriously. When designing a visual notation, one should take into account both its semantics (what is represented) and its syntax (how it is represented) to carry out the information in the best way possible. The *Physics of Notation* proposes a set of principles for designing effective visual notations, with a focus on syntax [41]. The *Guidelines of Modeling* offers a more general approach on designing information models [48]. To date, most software visualizations still revolve around understanding software through its structure, behavior and

evolution using hierarchical or graph visualizations [34].

This thesis is concerned with the software visualization of usage data. Limiting the analytics to usage data is interesting because it is not very much studied, compared to the other types of software data. Especially in software visualizations, PDD has been very little used. We will focus on the visualization and analytics part, detaching ourselves from the task of collecting the data itself. The rest of this chapter will describe how the topic came to be, and how we formed our research questions.

1.1 Defining Goals

This thesis was started with the general goal of working on software visualizations. Anna-Liisa Mattila ([34, 35, 36, 37]) has been working extensively on this topic, and it was decided to use her work as a basis. Using the Unified Data Model, the functioning database and visualization template described in [36] (cf. Section 2.5), we would try to extend the proof-of-concept by creating new visualizations for a different data set. An agreement with Esko Pekkarinen ([23]) was made, that we could use the data collected on the software Kactus2 (cf. Section 3.1). From this, we defined the research questions, described below.

1.2 Research Questions

Given the data and tools at our disposal (cf. Chapter 3), the following research question was defined: *Is the presented data model and visualization framework suitable for developing software usage visualizations?*

More precisely, we tried to answer the following questions:

RQ1 Can the Unified Data Model for Software Engineering be used to efficiently model software usage data?

RQ2 Can the framework presented be used to develop insightful visualizations of software usage data?

2 THEORETICAL BACKGROUND

This chapter presents the theoretical foundations of this thesis. Section 2.1 introduces the field of software analytics. Section 2.2 describes the type of data that this thesis is concerned with: post-deployment data, and section 2.3 presents the techniques of data collection related to post-deployment data in particular. Finally, section 2.4 presents the state-of-the-art of software visualizations, and visualizations of post-deployment data.

2.1 Software Analytics

2.1.1 Software and Analytics

Analytics is the use of analysis and data to make better (i.e., more rational) decisions. This discipline shifts the decision-making process from intuition to systematic reasoning. Buse and Zimmermann suggest that the software industry has many qualities that make it fit for analytics: *data-rich, labor intensive, timing dependent, dependent on consistency and control, dependent on distributed decision making, and low average success rate* [9]. As a response to the increasing amount of available software data, the industry developed *software analytics*, which is defined by Buse and Zimmermann as:

Analytics on software data for managers and software engineers with the aim of empowering software development individuals and teams to gain and share insight from their data to make better decisions. [10]

Today, the amount of data available for analysis requires that software analytics make use of automated tools (at least to some extent). Moreover, decision-making should be based on up to date information. Non-automated collection and analysis techniques are in this regard often too slow [39].

Analytics can be seen as a multi-layered process, from the measurement of raw data to insight, through quantitative and qualitative analysis. This was described by Kaushik for web analytics [24], but can be easily generalized to software analytics (cf. Figure 2.1).

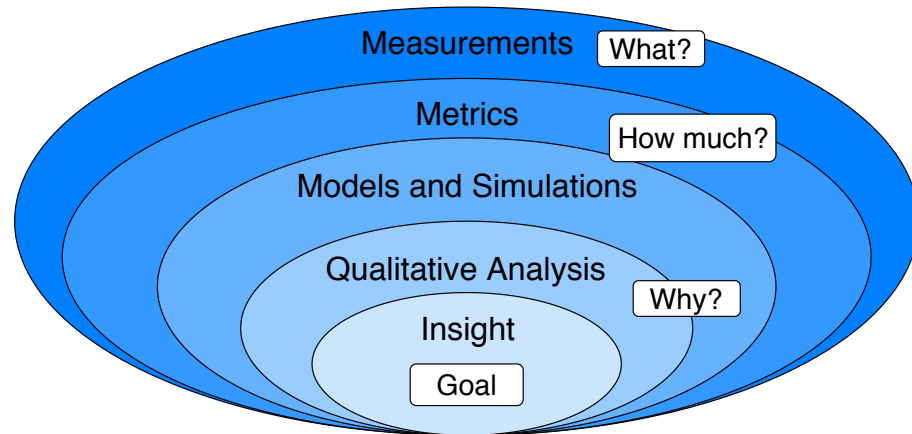


Figure 2.1. Overview of analytics, adapted from [24].

2.1.2 Types of Software Analytics

Software analytics can be classified in several ways. In [39], the authors detail different kinds of analytics. The following categories classify software analytics based on the team position, analysis quality, tool used, focus and target audience:

- *Internal vs External.* Analytics can be performed either by the internal team, or by an external entity. This brings two main challenges to the analytics. As an external team does not have access to live data, a copy must be sent outside a company's firewall for analytics. The other challenge is that such practices may require anonymization of the information to tackle privacy issues.
- *Quantitative vs Qualitative.* Quantitative methods rely extensively on automated statistical tools; qualitative methods are more manual and require more interaction. Both methods require careful design to be rigorous and thus useful.
- *Data mining tools vs Visualization tools.* Data mining tools are automatic processes; they are designed to produce one conclusion. Visualizations are interactive tools; they let users control the output of the analytics. The authors point out that most analytics today focus on data mining.
- *Explanatory vs Deployment.* An explanatory study is preliminary, and might or not be successful. In this kind of analytics, the goals (how to add value to the business) are not clearly defined yet. Deployment analytics is integrated throughout the years in the product life cycle. This second analytics is made possible only once the goals have been defined after a successful explanatory analytics.
- *Audience.* The potential audience for analytics: developers, managers, researchers, etc. Each type of audience having particular needs and wants.

Another way to classify analytics is with regard to its intent. Banerjee, Bandyopadhyay, and Acharya describe four types of analytics [2].

- *Descriptive* analytics answer the question of "What happened?" through collected

	Past	Present	Future
Information	What happened? (Reporting)	What is happening now? (Alerts)	What will happen? (Extrapolation)
Insight	How and why did it happen? (Modeling, experimental design)	What is the next best action? (Recommendation)	What is the best/worst that can happen? (Prediction, optimization, simulation)

Figure 2.2. Classification of analytical questions, from [12].

metrics.

- *Diagnostic* analytics focuses on understanding "Why?" something happened.
- *Predictive* analytics tries to forecast "What will happen?" based on the analysis of the past.
- *Prescriptive* analytics gives guidelines for "What should be done?" in the future.

Davenport, Harris, and Morison look at analytical questions that drive analytics. They divided questions into *insight*- and *information*- seeking, and whether they are concerned with the *past*, *present* or *future* (cf. Figure 2.2) [12]. Buse and Zimmermann report that insight questions are harder to answer than information questions, as well as questions regarding the future as compared to past or present questions [10].

2.1.3 Needs and Applications of Software Analytics

In practice, managers and developers use different software analytics with different objectives in mind. Buse and Zimmermann studied the needs of 110 professional developers and managers in software analytics. Their results show which metrics and artifacts are most important and used, and how analytics can help making better decisions [10].

Artifacts are the elements on which a metric is measured (for example, code complexity can be evaluated on a function, class, or entire product). Artifacts that were seen as important by developers or managers were: *feature*, *product*, *component*, *bug report*, *binary*, *test case*, *function*, *class*, *team*, and *author*. Analyzing individual features was considered to be the most important, but many artifacts are a source of unique information. Furthermore, it appears also relevant to consider these artifacts simultaneously, even though all are important by themselves.

Developers and managers use a great number of different metrics for analytics. According to [10], the most valued were *failure information* (feedback from end-users) and *bug*

reports (results of the testing processes). Buse and Zimmermann also asked what the participants of their study *would* use if it was available. An important result is that most metrics could be used by twice as many developers and managers, if these indicators were made available. Differences are noted between what is preferred by developers (*velocity, churn, readability, complexity*: code-related metrics) and managers (*failure information, telemetry, testing*: customer-related metrics). As with artifacts, there is a trend for using metrics together more than independently.

Understanding what information is truly needed requires as well understanding how this information can be used. The researchers of [10] grouped the answers of the participants around common themes. A variety of usages of software data was mapped out.

- *Targeting Testing*. Deciding which piece of code needs to be tested.
- *Targeting Refactoring*. Deciding which piece of code needs to be rewritten.
- *Release Planning*. Deciding which feature to release and when.
- *Understanding Customers*. Identifying which features are valuable or problematic for the users.
- *Judging Stability*. Anticipating future changes and deciding on the fate of a (sub)system.
- *Targeting Training*. Evaluating the training needs of new individuals or teams.
- *Targeting Inspection*. Deciding when to start a code review or inspection.

Among these, *targeting testing* was by far the most mentioned scenario by both developers and managers, followed by *targeting refactoring* (mainly developers) and *release planning* (managers).

2.2 Post-Deployment Data

2.2.1 Software Data

Software development produces a wide variety of data [9]. Whether it is the source code itself, the test logs or the bug databases, software development data is widely available, and has been very much studied [39]. But software data may also originate from outside its development period. One such data type is concerned with data produced after the product has been deployed: Post-Deployment Data (PDD), as described by Olsson and Bosch [43]. The authors identify five usages of PDD: *new feature development, feature improvement, feature usage, diagnostics, and operation/performance*. PDD itself comprises of a wide variety of data sources such as incident reports, usage logs, stack traces, etc.

Our thesis focuses on *usage data*, data originating from the interactions of the users with the software. This definition of usage data can be related to Schuur, Jansen, and Brinkkemper's definition of Software Operation Knowledge (SOK) usage data [49], with a

more precise focus on the user-interface interaction level. In [3], Begel and Zimmermann list 145 questions that software engineers would like data scientists to investigate. The top two questions of that list are related to software usage. This shows that there is a dire need of usage data for improving software products and processes.

In [55], Suonsyrjä et al. evaluate the potential of automatic collection of PDD to fulfill the needs for such knowledge in the Finnish software development industry. Post-deployment data is here used to consider data that is automatically collected after the commercial launch of a product. The results highlight that most of the different types of analyses desired from PDD were value analysis, users' problems detection, and use path analysis. Furthermore, the data types desired were time spent, performance, amount of use, and errors. Yet the main knowledge source were still revolving mainly around direct (meetings) or indirect (questionnaires) customer contact, which are not automated. Challenges mentioned in the study were broad, concerning acquiring and processing the data, as well as the maturity of processes and customers, and privacy concerns. Suonsyrjä et al. conclude that automatic collection of PDD has the potential to help teams improve their processes and products, but still lacks more research and development.

2.2.2 Applications of PDD

Post-deployment data has been used for several reasons in multiple contexts. In testing especially, usage data has been highly valued for some time. Brooks and Memon present a technique for testing GUI applications, based on usage profiles (sequences of events executed on a GUI, also called user-sessions) [5]. The web industry in particular has made a great use of PDD for automatic testing of web applications through user-sessions [14, 15, 47, 53]. Lautenschlager and Ciolkowski show that the collection of runtime data for incident diagnosis can be beneficial in cloud applications, but must be carefully designed to prove useful [27].

Web applications have also widely considered usage data for understanding the behaviour of users on certain platforms. Clickstreams (a user's path through a website) shows how a website is navigated. Such data has been used to study how a website is used, the propensity to stay or leave, etc [6, 28]. Atterer, Wnuk, and Schmidt further improve this idea by adding data such as mouse movements and keyboard inputs to better track user actions on web pages [1].

PDD has been used for identifying feature value through usage. Tyrväinen et al. use fast feedback from customers to improve the time needed for feature development [57]. Fabijan, Olsson, and Bosch study how feature value can be tracked, and how this metric can be used to make decisions for feature development or reduction [16]. Feature usage as a metric for feature reduction has also been studied by Marciuska, Gencel, and Abrahamsson [30, 31]. Mattila et al. use feature usage along issue management and development data in order to visualize usage density [35].

2.3 Data Collection

How data is collected is not necessarily obvious, and should not be disregarded. As this thesis is concerned solely with PDD, we will not elaborate on collection of other types of software data. In [56], Suonsyrjä et al. list and evaluate different technological approaches to usage data collecting.

2.3.1 Challenges and Limitations of PDD Collection

To evaluate the data collection methods, the authors of [56] build a list of the challenges and limitations of usage data collecting:

- *Number of use cases.* Too few use cases (i.e., planned concrete usages of the data) is an inefficient use of resources. Too many use cases may result in data collection that is not able to satisfy any of them.
- *Timeliness.* When information is generated must be considered, as it can limit the collection and use of the data. For example, incident reports are created only when incidents occur.
- *Confusion.* Continuous collection may incite to implement continuous testing (e.g., of partially developed interfaces). This can lead to user confusion.
- *Legality.* Country-specific laws, regulations and permissions must be taken into account when designing a usage data collection method. This is even more true since the recent application of the GDPR¹.
- *Safety.* The chosen technique must consider privacy and security issues.
- *Data sources.* If there are several distinct data sources, then either: the data must be unified before being processed (cf. Section 2.5), or the analyzer must be able to process the different types of data.
- *Lack of a systematic approach.* Without a systematic approach for collecting data, more difficulties in the course of the data-driven software development process will occur.
- *Usability.* The collected data should be easily available, attractive, and usable by the intended people.

These challenges and limitations will be considered in the selection processes of the right approach for a specific application.

¹<https://eugdpr.org/> - General Data Protection Regulation of the European Union

2.3.2 Technological Approaches

Suonsyrjä et al. identify five different approaches to usage data collection.

1. *Manual implementation.* Manual implementation is simply adding statements in the code, that will collect the desired data. Its advantages are that it provides very high flexibility and configurability while minimizing the overhead. The disadvantages of this approach are the big work effort required for large projects, and that it makes reuse almost impossible. Manual implementation is thus best suited for preliminary testing, or for a very specific data source and target.
2. *Automatic instrumenting with a separate tool.* Automatic instrumenting relies on tools (such as [11]) that can automatically insert instrumentation code into the source code. Compared to the manual implementation described above, this approach reduces greatly the efforts required, and provides much better reuse possibilities. On the other hand, these tools focus on only one source or target. This may raise the overhead proportionally. This approach is best suited in situations where the main criteria is the low implementation effort.
3. *Aspect-oriented approach.* The aspect-oriented approach has its roots in aspect-oriented programming (AOP), a programming paradigm that focuses on the modularization of concerns at the level of the source code. This method allows to consider complex conditions for executing the data collection code. This results in a very expressive technique that can target system- and application-specific instrumentation. Like manual implementation, it is very flexible. But as for the previous approach described, the automation of the instrumentation reduces greatly the work effort required. The possibility of reuse will be good only for applications that use the same syntax for the data collection.
4. *Alternative implementation of a UI library.* The modification of a standard UI library for user interaction allows to very easily collect usage data. The efforts required are thus low, but induce performance overhead. Reuse and configurability are not an issue. One concern is that some data might be out of reach from such UI libraries.
5. *Execution environment.* Without touching the software itself, the data collection can be done by the environment. It is especially the case for languages that are executed in a virtual machine, such as Java and JavaScript, where method and function calls can be monitored. The pros and cons are similar to those of the alternative UI library. One difference is that the data collected may require heavy post-processing.

With these five approaches for automatic data collection defined and their challenges and limitations, the authors developed a framework for selecting the proper approach.

2.3.3 Selecting the Right Approach

The authors describe in [56] already a first selection framework for automated usage data collection technologies. This framework was tested then refined in [54]. The version presented here is the improved one. Figure 2.3 summarizes the six steps of the framework.

1. *Defining a main goal* for the data collection. The target(s) must be specified in order to select the right approach.
2. *Removing irrelevant evaluation criteria* that do not apply to the project. A summary of these criteria and their relation to the collection techniques can be seen in table 2.1.
3. *Finding out the critical limitations* to the technological approaches, dependant on the context of the data collection.
4. *Rejecting the unsuitable techniques* if critical limitations are faced.
5. *Prioritizing the evaluation criteria* with regards to the main goal defined in the first step.
6. *Evaluating the remaining techniques*. The +/- indications in table 2.1 may serve as guidelines, but each project is different and must be treated with care.

This framework is intended as a general guide for companies and researchers wanting to implement a usage data collection system. See section 3.1.2 for details on how this framework was used to chose the data collection system for Kactus2.

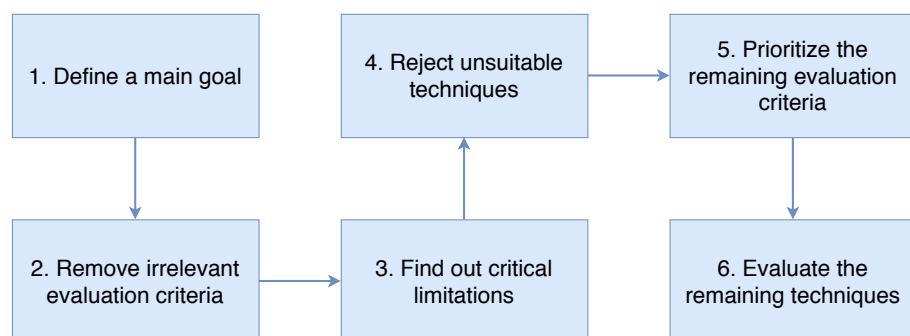


Figure 2.3. The refined collection framework for user-interaction collecting techniques, from [54].

2.4 Data Visualization

2.4.1 Designing Visual Notations

Ever since the beginning of the software engineering industry in the 1940s, data visualizations have played a major role. The first visual notation of software engineering data dating back to 1947 [19]. Today, software data visualizations are used at every stage

Table 2.1. Summary of the techniques evaluations, from [54].

Criteria	Techniques				
	Manual	Tools	AOP	UI Lib.	E.E.
Timeliness	+	+/-	+	+	+/-
Targets	+	-	+!	-	+/-
Scalability	-!	+	+!	+	+!
Overhead	+	-	+/-	-	-
Sources	+	-	-	-	-
Configurability	+	-	+	+	+/-!
Security	+	+/-!	+/-	+	+/-
Reuse	-	+	-	-	+!
Change	+!	+	-!	-!	+

+ = Supports selecting

- = Technique has limitations

+/- = No clear support nor limitations

! = A possible ground rejection

of the SE process. The reason for that is obvious: visual notations are extremely effective at sharing information, especially between people with different levels of technical knowledge.

In *The "Physics" of Notations* (...) [41], Moody defines a **visual notation** (what is called a *visualization* throughout this thesis) as: a set of graphical symbols (the **vocabulary**), a set of compositional rules (the **grammar**), and a set of definitions of the meaning of each symbol (the **semantics**). Together, the vocabulary and grammar form the **syntax** of the notation [41]. The author notes that while the semantics of notations in SE have been largely studied, the syntax was mostly left out of most evaluation methods. This in spite of the fact that the form (hence the syntax) of the visualization has been shown to be at least as important as their content, especially for beginners.

For a visual notation to be worth ten thousand words, it needs to be designed to achieve cognitive effectiveness [26]. This measure of how well a visualization works is based on the speed, ease, and accuracy required by the human brain to process it. For evaluating the visual notations, the widely accepted method has for the most part been *ontological analysis*: a two-way mapping between the notation and the ontology (cf. Figure 2.4). The problem of the ontological approach is that it targets the semantics of the notation, but not its syntax (*what* is represented: the concept-construct mapping, not the form that the constructs take).

To fill the need for evaluating the syntax of visual notations, Moody proposes a **descriptive theory** of how visual notations communicate, and a **prescriptive theory** of principles for designing effective visual notations [41]. The communication theory is represented in figure 2.5. The sender encodes the message in the form of a diagram (using a visual no-

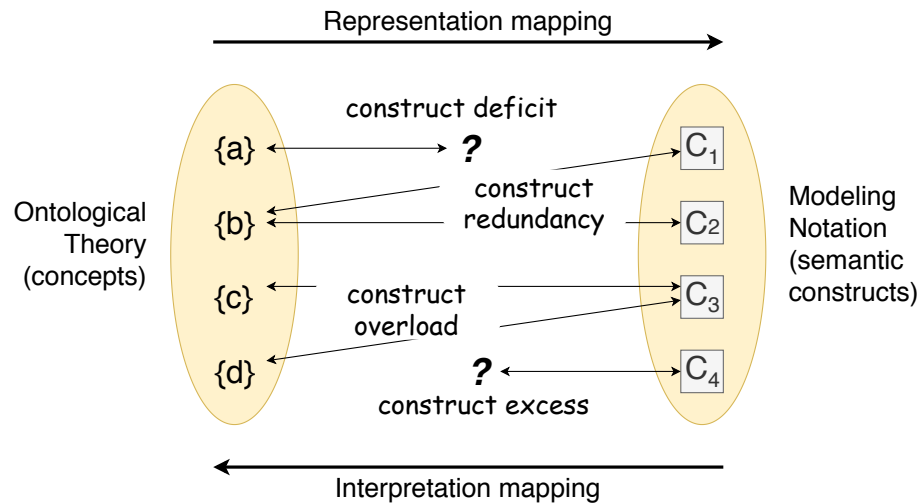


Figure 2.4. Ontological mapping, from [41]. There should be a 1:1 mapping between the concepts and the constructs to avoid construct deficit, redundancy, overload, and excess.

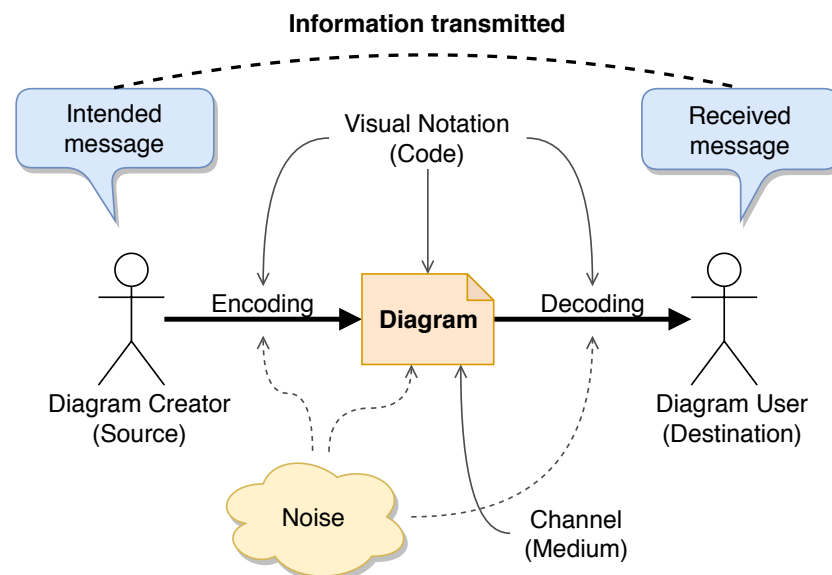


Figure 2.5. Descriptive theory, from [41].

tation), and the user decodes this signal. The medium is the physical platform on which the diagram is shown, and noise represents the interference that can happen at every level. *Effectiveness* is here measured by the match between what the creator intended and what the user actually received.

The prescriptive theory describes 9 principles for designing effective visual notations (cf. Figure 2.6).

1. Semiotic Clarity This principle is an extension of the aforementioned (and detailed later on) ontological analysis. Based on the *theory of symbols* of Goodman [20], it states that there should be a one-to-one correspondence between the symbols and their referent concept. Applied to visual notation, we are looking at a mapping of the *semantic*

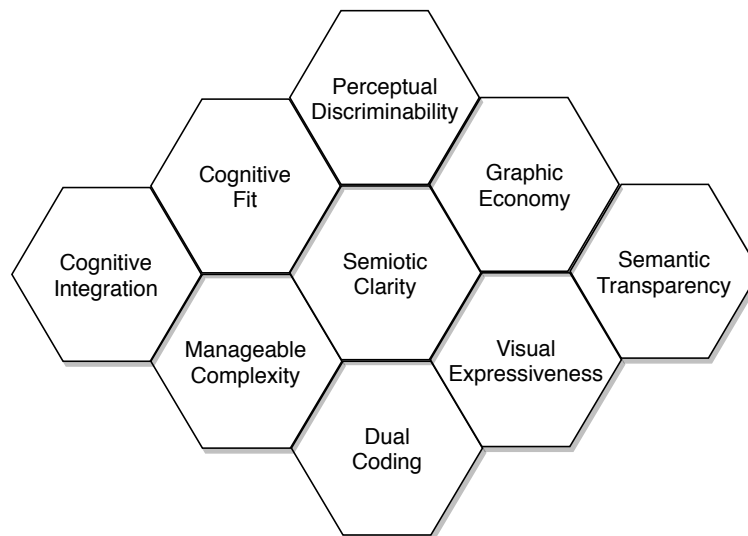


Figure 2.6. Prescriptive theory principles, from [41]. The honeycomb structure illustrates the desire of the authors to encourage modifications and extensions to the principles by further research.

constructs and the *graphical symbols* of the visual notation. When the one-to-one correspondence is not achieved, the following phenomenon are observed. Figure 2.7 shows the four phenomenon undermining semiotic clarity.

- *Symbol redundancy* occurs when multiple graphical symbols have the same semantic meaning, they can be interchanged without modifying the meaning of the notation. Such symbols are called **synographs**. This adds an extra cost to the user of the visualization to remember several symbols for the same construct and should be avoided.
- *Symbol overload* is the situation where a single graphical symbol does not have a unique meaning. Such symbols are called **homographs**. As they introduce ambiguity within the notation, they should as well be avoided.
- *Symbol deficit* happens when some semantic constructs are not represented by any graphical symbol. While this undermines semiotic clarity, it is not always a bad thing in SE visual notation which are often highly complex. In this regard, not showing all constructs on a visualization helps reduce complexity, improving understanding.
- *Symbol excess* on the contrary, occurs when graphical symbols bear no meaning: they are not linked to any construct. Such as using explicit symbols to display *comments*. As they increase the complexity without bringing any additional information, they should be avoided.

2. Perceptual Discriminability The accuracy with which one can distinguish the symbols is crucial for the accurate interpretation of a diagram [63]. Perceptual discrimination is the first phase of *perceptual processing* (i.e., seeing): the automatic, very fast process that occurs before the actual cognitive tasks can start [42].

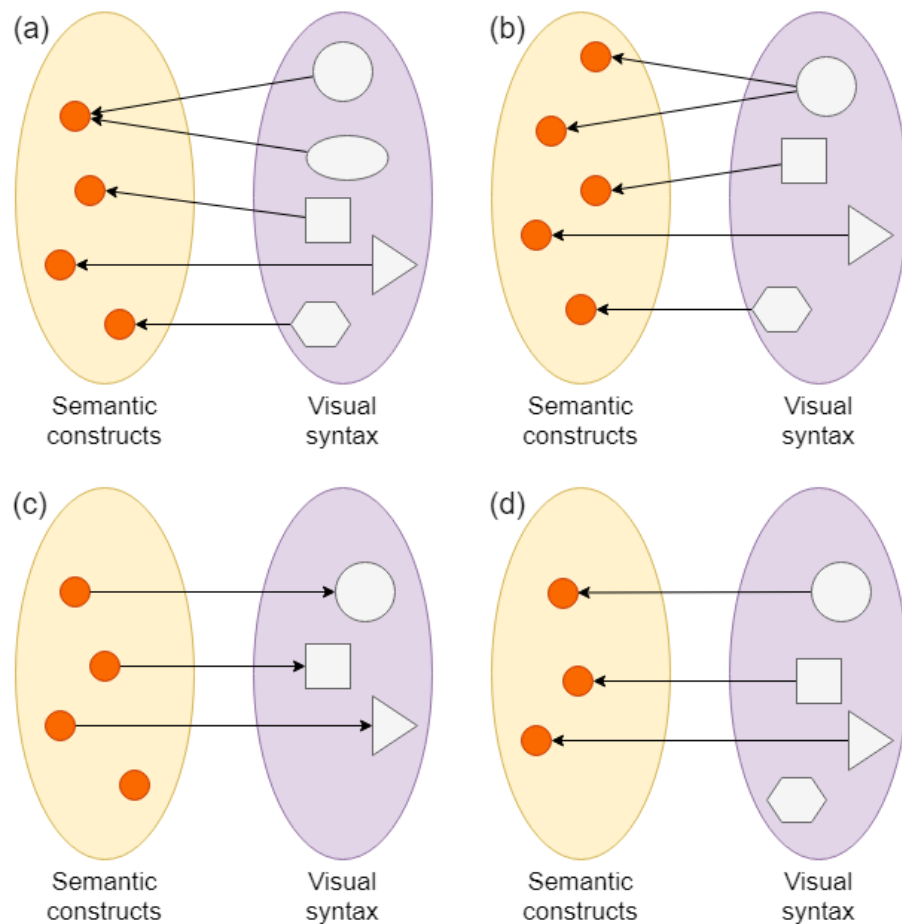


Figure 2.7. *Semiotic (non-)clarity: (a) symbol redundancy, (b) symbol overload, (c) symbol deficit, (d) symbol excess.*

- *Visual distance* is the primary determinant of discriminability between symbols. We measure it by the number of visual variables that differ and the size of these differences. The 8 visual variables (cf. Figure 2.9) were defined by Bertin, Berg, and Wainer [4].
- *The primacy of shapes* states that the shapes of graphical symbols is the primary basis for distinguishing between them. Hence the size of the repertoire of shapes used in a visual notation should be a primary concern.
- *Redundant coding* of information reduces the risk of misinterpretation and diminishes the effects of noise. Using several visual variables to identify unique symbols is a simple example of redundant coding.
- *Perceptual popout* happens when visual elements have a unique value for at least one variable. They 'pop-out', meaning that they can be detected and processed in parallel. This increases the cognitive efficiency of the diagram.
- *Textual differentiation* of symbols is an ineffective way of dealing with complexity. It relies on adding text to discriminate between otherwise identical graphical symbols. While this can be used to distinguish symbol *instances*, it should not be used to distinguish symbol *types*.

3. Semantic Transparency This principle requires that graphical symbols provide cues to their meaning. Semiotic transparency is a continuum between directly perceivable and opaque meaning. This concept (here described for the generic notion of *symbol*) applies to both icons and relationships in a visual notation.

- *Semantically immediate* symbols can be directly understood even by a novice simply through its appearance.
- *Semantically opaque* symbols have a purely arbitrary relationship between the symbol and its meaning.
- *Semantically perverse* symbols can induce a novice to infer a different or opposite meaning based on the appearance of the symbol.
- *Semantically transparent* symbols provide a cue to the meaning, but require some initial explanation.

Perceptual resemblance, having common logical properties, functional similarities, or using metaphors and cultural associations help with semantic transparency.

4. Complexity Management This principle reflects the ability to deal with complexity (i.e., the amount of elements on a diagram) without overloading the human mind. The limits of the human mind are both perceptual (the perceptual discriminability capabilities) and cognitive (working-memory capacity). Two main techniques can be used to handle complexity.

- *Modularization* consists of reducing the complexity of large diagrams by dividing them into subsystems. This reduces the *cognitive load* and helps increase the speed and accuracy of understanding information [38]. Semantic constructs (like subsystems) are not enough on their own; they need to be supported by diagrammatic conventions as well (syntactic level).
- *Hierarchy* is a very effective way of making complexity manageable by humans. The top-down construction of diagrams is called *decomposition* or *refinement*, while a bottom-up approach is called *abstraction* or *summarisation*.

5. Cognitive Integration This theory applies when multiple diagrams are used to represent different aspects of a system. Whether it is diagrams of the same type that apply to different parts of the system (i.e., **homogeneous integration**), or different types of diagrams (i.e., **heterogeneous integration**). The **cognitive integration of diagrams** theory states that multi-diagrams representation must support both *conceptual and perceptual integration* to be cognitively effective (figure 2.8) [25].

- *Conceptual integration*: mechanisms that help the user assemble information from several diagrams into one coherent representation. **Summary diagrams** (that provide a simplified view of the whole system) and **contextualization** (showing the

focus of the current visualization in the context of the system as a whole) are two effective techniques.

- *Perceptual integration*: indicators to simplify the transition and navigation between the diagrams. Many mechanisms can help perceptual integration, such as **identification** (i.e., clear labelling of diagrams), **level numbering**, including **navigational cues** or even a **navigational map**.

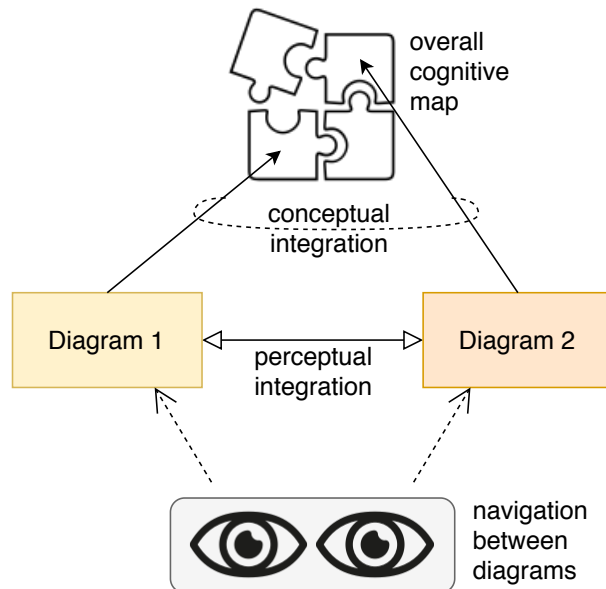


Figure 2.8. Cognitive integration: multiple diagrams are used to represent a system.

6. Visual Expressiveness This is a measure of the number of visual variables used in a notation. While the perceptual discriminability describes the visual variations between two graphical symbols, visual expressiveness measures the utilization of the graphic design space. **Information-carrying variables** are variables used to encode information, and **free variables** are variables that are not formally used in the notation. The **degree of visual freedom** is the number of *free variables* used, and is the inverse of the visual expressiveness. Textual notations have 8 degrees of visual freedom (they are called **non-visual**); a notation that has 0 degree of visual freedom is **visually saturated**.

While color is one of the most effective visual variables, it should not be used alone but in conjunction with others. The choice of the variables used should be based on the nature of the information: *form follows content*. The aim is to match the properties of the visual variable with those of the construct to be represented. Figure 2.9 shows the 8 visual variables as defined by Bertin, Berg, and Wainer [4].

7. Dual Coding This theory advises to use both text and graphics at the same time to convey information more efficiently than if either was used on their own [45]. Even though just like color (and we have discussed this before are well), text should not be used on

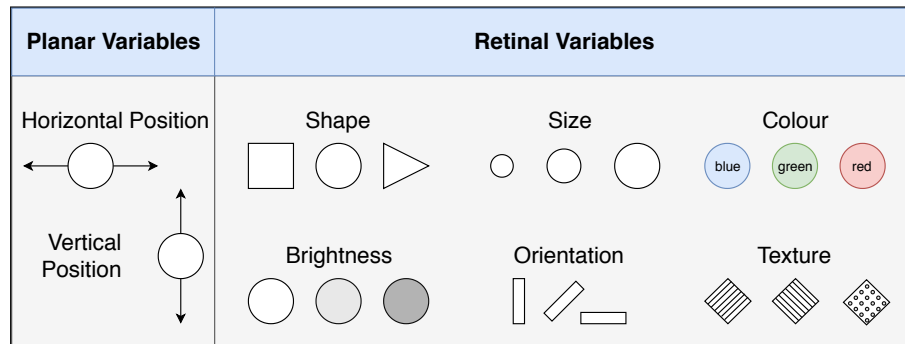


Figure 2.9. The 8 visual variables, from [4].

its own to distinguish between two symbols, it can be a form of redundant coding. **Annotations** directly of the visual notation can be used to clarify meaning and improve the understanding of the diagram. Another application of dual coding are **hybrid symbols**: a text + graphics combination that reinforce or extend the meaning of the graphics (cf. Figure 2.10).

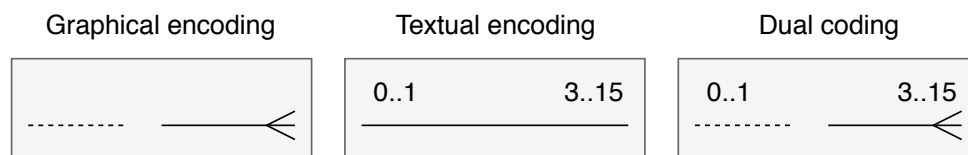


Figure 2.10. Hybrid symbols: implementing dual coding by using a combination of text and graphics.

8. Graphic Economy The number of different graphical symbols used in a notation (**graphic complexity**) should be cognitively manageable. The graphic complexity of a notation is measured by the size of its visual vocabulary. While diagrammatic complexity (cf. Principle 8: Complexity Management) looks at the number of *elements* displayed, graphic complexity looks at the amount of *different symbols*. The **span of absolute judgement** of an individual - his ability to distinguish perceptually distinct alternatives - is around 6 categories [40]. Therefore, this is our upper limit for graphic complexity. Graphic economy can be achieved in different ways.

- *Reducing semantic complexity* by removing or partitioning the spectrum of semantic constructs.
- *Introducing symbol deficit* by choosing not to show some semantic constructs graphically and introducing more textual encoding.
- *Increasing visual expressiveness* by increasing the number of visual variables used to discriminate between symbols. Indeed, the limit of 6 symbols applies only if a single visual variable is used for perceptual discrimination.

9. Cognitive Fit The use of different visual dialects for different tasks and audiences is a principle of **cognitive fit theory** that is widely accepted in the IS field [50, 59, 60]. To

achieve good performances in problem-solving, there should be a three-way fit between *the problem representation, task characteristics, and problem solver skills*. This implies that different visual notations should be used for different combinations of these three elements. Two main reasons drive the need for the use of multiple notations regarding cognitive fit.

- *Expert-novices differences* suggest that novices will require visualisations that have better perceptual discriminability, reduced complexity, more semantic transparency, use explanatory text, and present simplified vocabularies than experts.
- *The representational medium* is a major determinant in the design of a visual notation. Whether a diagram will need to be drawn by hand, or is solely generated by software should impact the requirements of the notation.

Interactions between the principles The interactions that these nine principles have among each other is shown in figure 2.11. Some principles work well together, creating synergies. Others have a negative influence on each other, and trade-offs must be made. Note that all relations are symmetric. Following are the main interactions:

- *Semiotic Clarity*. Symbol excess and redundancy increase *Graphic Complexity* (negative) while symbol overload and deficit reduce it (positive).
- *Perceptual Discriminability*. Increases *Visual Expressiveness* (due to the increase in visual distance). The reverse is also true.
- *Visual Expressiveness*. Reduces the effects of *Graphic Complexity*. *Graphic Economy* limits the use of *Visual Expressiveness*.
- *Graphic Economy*. A high amount of symbols decreases *Perceptual Discriminability* by making it more difficult to distinguish them.
- *Cognitive Fit* is influenced positively by *Perceptual Discriminability*, *Complexity Management*, *Semantic Transparency*, *Graphic Economy*, and *Dual Coding* by helping novices. *Semantic Transparency* may decrease *Cognitive Fit* fit for experts.
- *Cognitive Fit*. *Semantic Transparency* and *Visual Expressiveness* both make hand drawing more difficult.

Eight years after [41], Van Der Linden and Hadar discuss the application of the PoN (the prescriptive theory) in their systematic literature review [58]. In their analysis of 70 applications of the PoN, a little more than half concerned the creation of new notations, the other half being existing notations or forks. In most cases, no other notation than PoN was considered. More surprisingly, the users of the notation were not involved at all in the vast majority of analyses. Trade-offs in the notation were not discussed with users, and proposed design changes or new notations were not evaluated on their cognitive effectiveness. This shows that even though PoN is highly used for analysing and designing visual notations, this is not done with much care. The core idea of PoN that designing visual notations should be a rational process, not a subjective one, seems to often be left out.

	Semiotic Clarity	Perceptual Discriminability	Semantic Transparency	Complexity Management	Cognitive Integration	Visual Expressiveness	Dual Coding	Graphic Economy	Cognitive Fit
Semiotic Clarity									+/-
Perceptual Discriminability					+				+
Semantic Transparency	+								+/-
Complexity Management								-	+
Cognitive Integration	-		+					-	
Visual Expressiveness	+							+	+/-
Dual Coding									+
Graphic Economy	+	+			-				+
Cognitive Fit									

Figure 2.11. Interactions between the principles, from [41]. + indicates a positive effect, - a negative effect, and +/- means the effect is either positive or negative depending on the situation.

2.4.2 Designing Information Models

While the PoN focuses on visual notations in the domain of SE, a more general model is the *Guidelines of Modeling* (GoM) from Schuette and Rotthowe [48]. This approach proposes a framework for rational design of information models. Six main principles are described:

1. **Construction Adequacy.** A consensus (between the model designers and users) must be found on the represented problem and on the model. Without this, the designer can not ensure that the construction quality of his model can be evaluated.
2. **Language Adequacy.** The language used to construct the model (e.g., a visual notation) must be suitable and correct. Suitability will be impacted by the choice of the modeling technique and relevant model constructs. Correctness refers to the proper application of the meta-model's grammar and syntax.
3. **Economic Efficiency.** This principle emerges from a purely economic yet rational point of view, where resources are not unlimited. The benefits coming from the designed model must outweigh the costs of developing it. Note that this principle may conflict with other principles.
4. **Clarity.** The model must be easily comprehensible for the user. Hierarchical decomposition, layout design and information filtering should be done by the model with the user in mind. Filters may be content-specific (show different detail levels in one model) or methodical (which allow the user to configure the meta-model).
5. **Systematic Design.** This expresses the need for an inter-model consistency between structural models (logical composition of structure) and behavioural mod-

els. A well-thought meta-model of information systems should indeed include both types.

6. **Comparability.** Models and meta-models should be comparable semantically regarding their correspondence and similarity. Two models can only be compared if each element of one model can be expressed in the other model.

The GoM-Architecture further specifies that these guidelines can be used independently (or with varying importance), depending on the restrictions of the actual modeling project.

2.4.3 Semantic Analysis of Visual Notations

As previously discussed, *ontological analysis* (or semantic analysis) is now a widely accepted method for evaluating visual notations, especially in the field of software engineering [17, 51]. A typical approach is to base the analysis on the Bunge-Wand-Weber (BWW) model of information system [44]. The BBW model (cf. [61, 62]) is derived from Mario Bunge's ontological model, which is inspired by systems theory [7, 8].

Ontological analysis postulates that there should be a bijection (i.e., two-way mapping) between the concepts of the ontology, and the semantic constructs used in the model. In this context, an ontological concept is a real-world construct (e.g., *thing*, *property*, *system*). Semantic constructs are obtained from a description of the grammar (here, the grammar is a visual notation). The first mapping - from concepts to constructs - is called the *representation mapping*. This mapping describes how the ontological concepts are represented by the semantic objects. The second mapping - from constructs to concepts - is the *interpretation mapping*. This mapping describes whether and how a semantic construct stands for in the ontology. The representation mapping brings the notion of *completeness*; achieved when the representation mapping is *total* (i.e., when all concepts can be represented by constructs). With the interpretation mapping comes on top the notion of *clarity*; achieved when the interpretation is both *total* and *one-to-one*.

A visualization will be said to be *complete* if the representation mapping is total (all concepts can be represented by constructs). Otherwise, the visualization is said to be *incomplete* or to have *construct deficit* (cf. Figure 2.12). A construct deficit means that not all ontological concepts can be represented, which is usually undesirable.

The ontological clarity of the visualization - how 'clearly' each construct represents a concept - can be undermined by *construct overload*, *construct redundancy*, and *construct excess*. These can be seen through the interpretation mapping (cf. Figure 2.13). Construct overload appears when one construct maps into one or more concepts. Construct redundancy shows that several constructs map to the same concept. Finally, construct excess arises when a construct does not map to any concept (this exposes a deficiency in the ontological model chosen, or that the construct goes beyond the intended scope of the visualization).

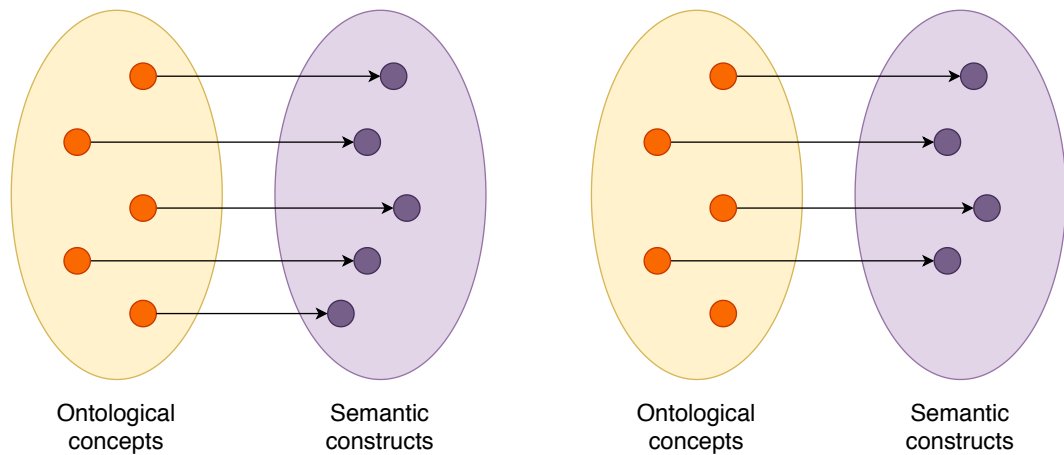


Figure 2.12. *Ontological completeness (left) and ontological incompleteness or construct deficit (right).*

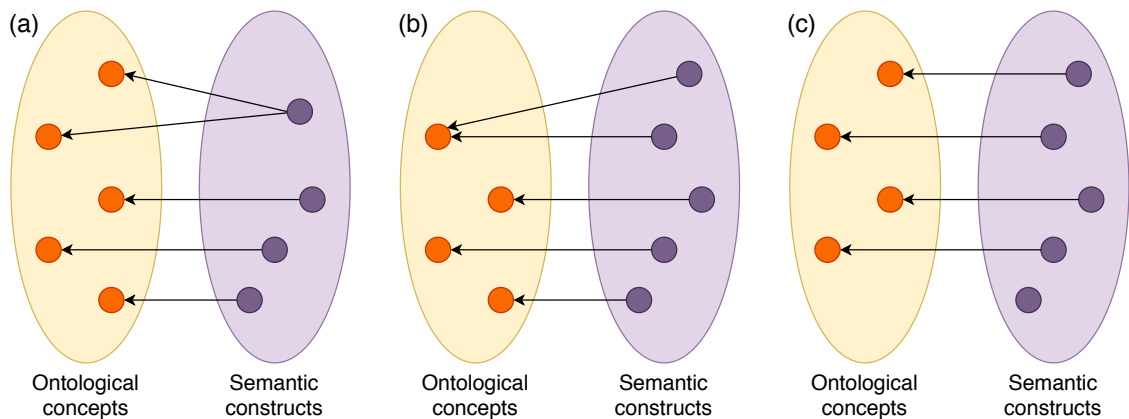


Figure 2.13. *Ontological completeness with from left to right: (a) construct overload, (b) construct redundancy, (c) construct excess.*

The *expressive power* of a visual notation is measured by both its ontological completeness and clarity. This notion can be used to compare two versions of the same visualization, for example. Assuming that the chosen ontological model was valid from the start, the evaluation of a visualization's completeness and clarity can indicate potential improvement areas.

The ontological concepts of the BWW-model are detailed in appendix B.

2.4.4 Software Visualization Today

In their recent systematic literature review (SLR), Mattila et al. describe the state of the research on software visualization [34]. Diehl defines software visualization as *visualizing the structure, behaviour, and evolution of software* [13]. But as the authors of the SLR explain, SE processes generate data (such as development data) that does not fit into that definition. In the SLR, the authors tried to understand 1) The reasons, methods, and

Table 2.2. Studied aspects in software visualization research, from [34].

Studied aspects	Structure	Execution	Evolution	Management	Development	Requirements	Optimization	Rendering	Other	
Amount of studies	37	25	15	6	3	3	3	3	2	
Data source x studied aspects	Structure	Execution	Evolution	Management	Development	RE	Optimization	Rendering	Other	Total
Source code	33	6	11	5	0	1	0	2	0	46
Software execution data	7	22	1	0	0	0	2	1	1	28
Change / version data	5	1	10	3	2	0	1	0	1	17
Static code analysis	8	5	2	1	0	0	2	0	0	15
Usage data	1	2	2	0	1	0	0	0	0	5
Documents and models	2	0	0	0	0	3	0	0	2	5
Test data	1	2	1	1	0	0	0	0	0	4
Other	2	1	1	0	0	0	0	0	0	4
Not relevant / Not stated clearly	4	0	3	2	1	0	0	2	0	9
Visualization format x studied aspects	Structure	Execution	Evolution	Management	Development	Requirements	Optimization	Rendering	Other	Total
Hierarchical and Graph-Based Techniques	31	17	11	3	2	2	1	3	2	61
Geometric projection techniques	12	12	5	2	0	0	2	1	0	26
Timelines	1	9	5	3	3	0	1	0	0	18
Info graphics	7	6	4	3	1	0	1	0	0	17
Icon-based techniques	6	3	4	0	0	0	1	0	0	12
Text based visualizations	5	1	2	0	0	0	0	0	0	7
Tag- and word-clouds	2	0	3	1	0	0	0	1	0	5
Pixel-oriented techniques	1	3	0	0	0	0	0	0	0	4
Other	3	4	4	3	0	1	1	0	0	11
Not stated clearly	1	0	0	0	0	0	0	0	0	1

data sources for software visualization, and 2) The maturity of the field of research.

The results show that the reasons for using software visualizations are mainly *program comprehension, collaboration and engagement*, as well as *support and maintenance*. What is generally visualized is the *structure and execution* of the software, and several *code quality metrics*. Most visualizations were developed to be used by *developers* and *software architects*, few were intended for *testers* or *managers*. The methods most used for the visualizations were *graph and tree visualizations*. Others include *timelines, poly-metric views, clusters, and heat maps*.

Based on the studied themes, data sources and visualization methods, the authors classify the 83 articles reviewed into 9 categories. Understanding 1) software structure, 2) software execution, and 3) software development. Showing 4) the evolution of software, 5) software project management. As well as 6) requirements engineering, 7) optimization of computing and resources, 8) visualization algorithms and rendering, and 9) others. The results show that the theme studied influences the data sources and methods used for the visualization (cf. Table 2.2).

This systematic literature review shows that today, software visualizations are still mainly used for understanding software structure, behavior and evolution through hierarchical or graph visualizations, by analysing source code. Moreover, while the field of research has developed quite well and is very active, it is still lacking maturity.

2.4.5 Usage Data Visualization

The main propose of PDD visualizations is currently related to feature usage. Mattila et al. integrated usage data along with issue management data and development data to create a mashup visualization [35]. The aim of their visualization was to show how

well continuous delivery of features was achieved through the project development. The timeline-based visualization (cf. Figure 2.14) is divided in two parts. The upper part shows features' lifespans (1 feature per line) from *issue created* (in blue), then *development on-going* (in yellow), to *development done* (in gray). The pink vertical lines following the end of the development show the usage of the feature by the end-users. The bottom part of the visualization shows the amount of *unfinished* (yellow) and *finished* (gray) features.

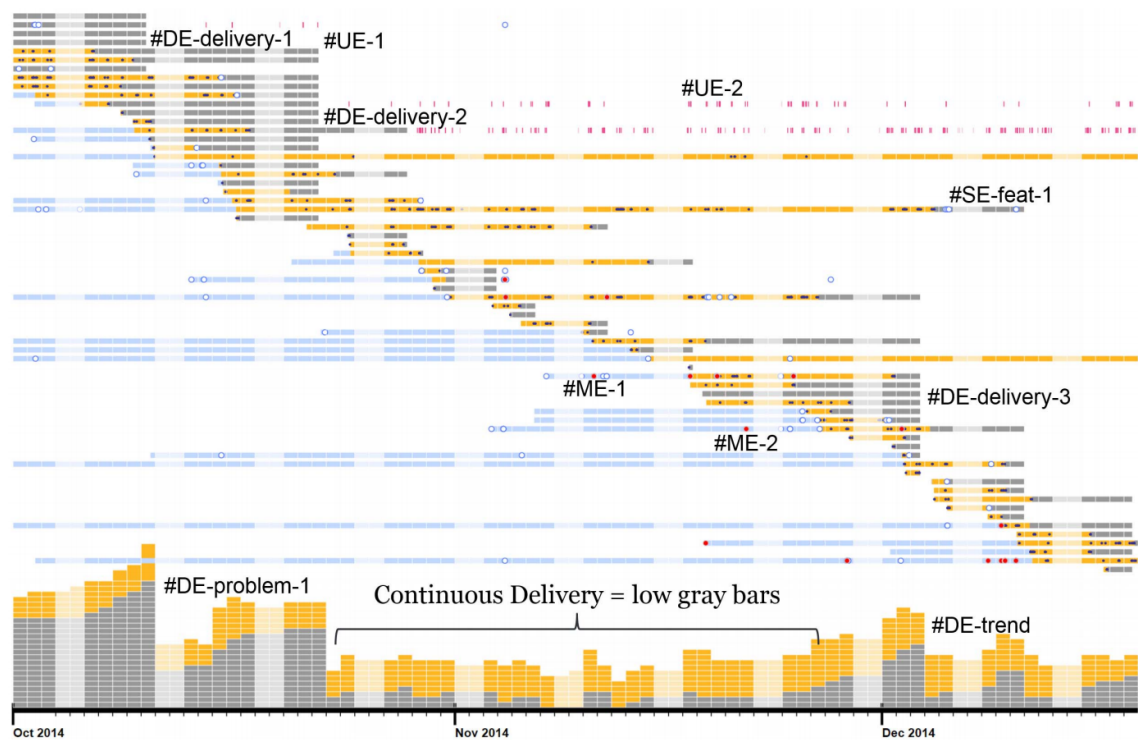


Figure 2.14. Mash-up software visualization, from [35].

Marcuska et al. make use of usage data to generate a *Feature Usage Diagram* [32]. Their work focuses on the usage of features per users (which features was used by whom and when), in the context of feature-reduction: selecting which features to remove in a (possibly bloated) software. Their visualization (cf. Figure 2.15) is a directed graph linking features together, and indicating their usage. Links indicate that a feature can be accessed through another feature. Feature groups represent sets of features that share access to the features outside the group. Marcuska and Abrahamsson present a visualization tool: *Feature Usage Explorer*, that implements this visual notation for HTML5-based applications [29].

Matejka, Grossman, and Fitzmaurice present a usage collection and visualization tool for generating heat-maps of GUIs [33]. The colored heat-map displays the usage of features in an interface, directly on top of it. Figure 2.16 shows the result of a heat-map on a classical GUI (Microsoft Word in this case).

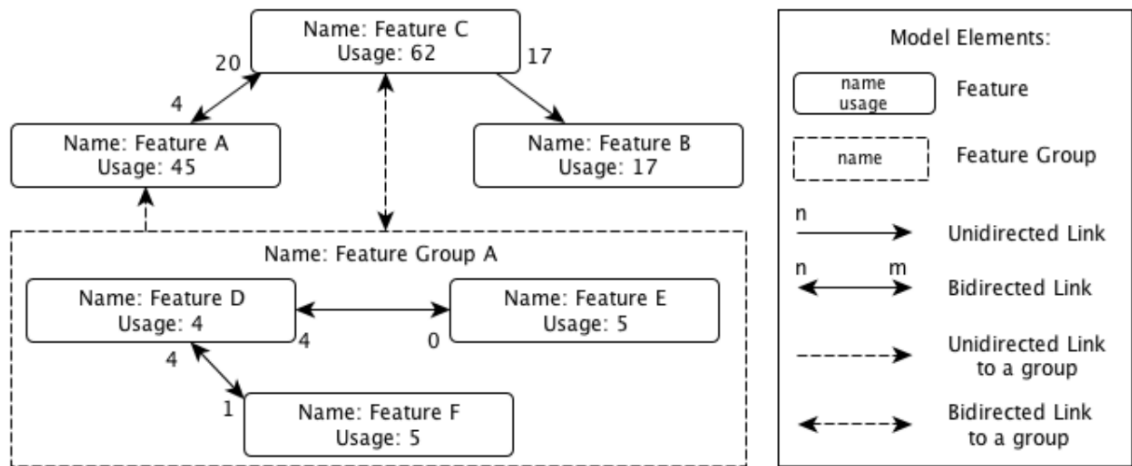


Figure 2.15. Feature Usage Diagram, from [32].

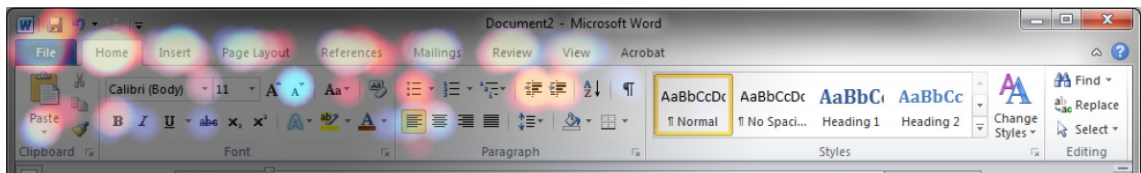


Figure 2.16. UI heat-map, from [33].

2.5 Data Modeling

From the data source to the data visualization, a necessary step is the use of a data model to represent and store the data. In [36], Mattila et al. present the **Unified Model for Software Engineering Data** (UMSED). The objective of this data model was to overcome the barrier of having multiple data formats (resulting from multiple data collecting tools and software engineering processes). It does so by defining a common format for software engineering data that can therefore be used as a basis for building reusable visualization and analysis components.

The version of the model described here is an adaptation that differs from the original. This version refines the notions of *Event*, *StateChange*, *Construct*, and *Origin*. How these elements are linked to each other can be seen in figure 2.17.

Event is an action performed by a user on one or more constructs at a certain time.

Construct (or artifact) is an aspect of software engineering that is interesting for visualization and analysis purposes. Constructs are the objects on which events are performed, and can also be the user himself.

StateChange is an optional attribute of an event. It is used when the event triggers a change of state in the construct it is linked to.

Origin is the source and context of the data.

This model allows to represent software engineering data in a generic way. Several data

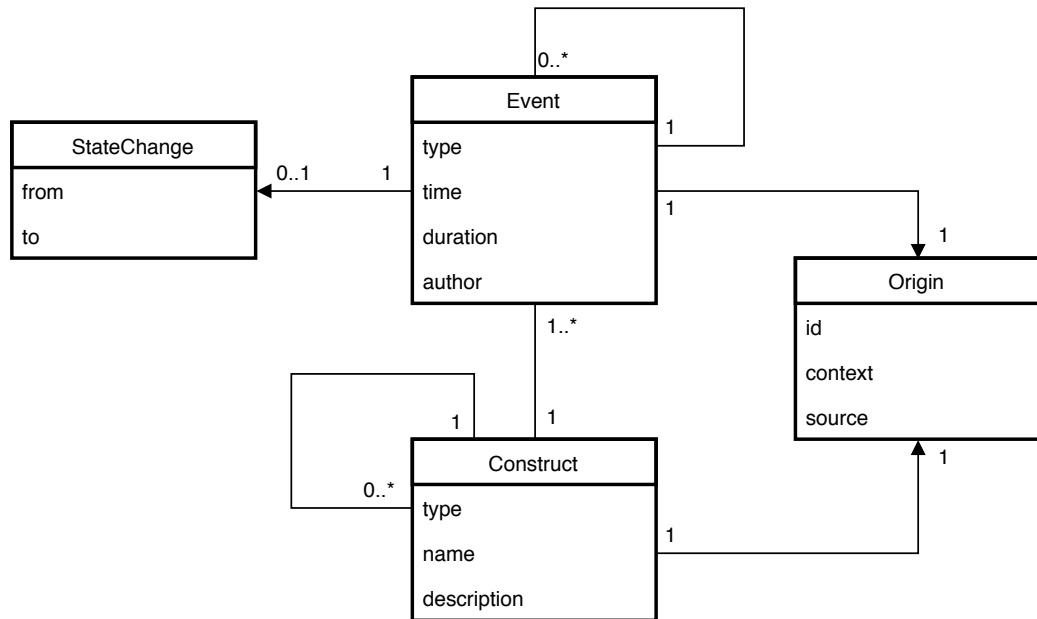


Figure 2.17. Graphical representation of the Unified Model for Software Engineering Data, from [36] (updated).

types originating from different data sources can here be easily linked to each other, once converted to the data model.

3 RESEARCH MATERIAL

This chapter presents the research materials that this thesis is based on. Section 3.1 presents the data that is used: its source (the software Kactus2), how that data had been collected, and the analytics that were previously made on this data. Section 3.2 describes the visualization framework in which we have integrated the data, as well as the original visualization plugin, and previous applications of these tools.

3.1 Data Source

3.1.1 Kactus2

The data that serves as the material for this thesis originates from Kactus2. This software is being developed at Tampere University by a research group of the Pervasive Computing Department. It was first presented in [22] and published as open source in 2011. Its source code can be found on github¹.

The development team describes the software as "a graphical EDA toolset for designing embedded products, especially FPGA-based MP-SoCs. The toolset supports reuse, exchange, and integration of Intellectual Properties (IPs), thus increasing productivity of hardware development, but potentially also assisting co-operation with software development."² EDA (Electronic Design Automation), also called Electronic Computer-Aided Design (ECAD), is a category of software tools for designing electronic systems. An MP-SoC (Multiprocessor System on a Chip) is an integrated circuit that integrates all components of a computer or other electronic system that has the particularity of having several processors.

The target users of Kactus2 are SMEs (Small and Medium-size Enterprises) that use FPGAs and on- or off-chip processors. The software is also used by students, for example at Tampere University (as it will be discussed later). The latest version of the software (Kactus2 3.6.5) was released on 29/06/2018³ for both Windows and Linux operating systems, and has been downloaded more than 500 times to date. It is written in C++11 and Qt 5.10.1, and currently contains more than 300,000 lines of code.

¹<https://github.com/kactus2/kactus2dev>

²<http://funbase.cs.tut.fi/#kactus2>

³<https://sourceforge.net/projects/kactus2>

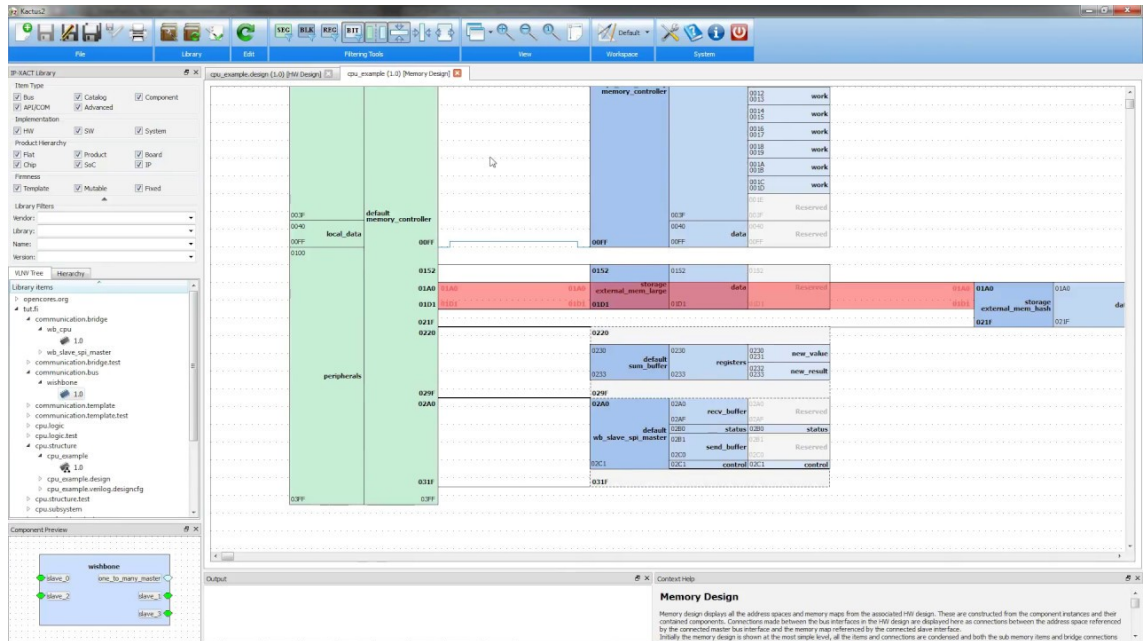


Figure 3.1. An example of the Kactus2 Interface.

3.1.2 Data Collection

The data was collected in the context of [54]. In this paper, Suonsyrjä describes the reasons for choosing specific collecting techniques in 3 case study companies. With the help of the author, the development team of one case study (Kactus2) used the framework designed in [56] to select the appropriate data collection approach. After evaluating several options and their feasibility, the selected technique was the *aspect-oriented approach* (cf. Section 2.3.2). The most important criteria considered were: security, amount of sources and targets, configurability, low work effort and reuse possibilities.

The data that serves as our research material was collected in the beginning of 2017, on a group of students in several laboratory exercise sessions. The collected data was stored in a log file for each user session of Kactus2. The logs were then consolidated into a unique file for analysis. The log consists of around 4,500⁴ timestamped entries recorded by the collection system.

```
25.01.2017 08:32:24 user X in session Y: Document Z unlocked.
```

Listing 3.1. Example of a log entry.

Each line of the log consists of the following information (cf. Listing 3.1):

1. The `user_id` of the user.
2. The `session_id` of the recorded session. Note that a session cannot be restarted once it has been quit.

⁴Due to the way the collection was implemented, some events were recorded twice. Other anomalies in the logs required corrections.

3. The `time` and `date` of the recording.
4. The `action` that was recorded. The complete list of all different actions present in the logs can be seen in appendix A.

The data was collected to answer three different questions:

1. *What features are used/not used?* This question was deemed too general and was narrowed thus down to: *Which toolbar actions are used most?*
2. *How long does it take for the user to find the unlock button?*
3. *Which help pages are viewed most?*

To answer the first question, all *toolbar button presses* were recorded. The objective was to count the number (or percentage) of uses of these features, in order to possibly rearrange the feature icons accordingly. To answer the second question, the timestamps of when documents are opened, closed, locked or unlocked were recorded. The researchers and developers wanted to achieve multiple things: finding out if the app was intuitive (especially for the first time users), profiling users based on their level, and investigate a potential problem with the unlock-feature. To answer the third question, the collection system records the switch from one interface to the other. This is because a help page is automatically opened as a sidebar by Kactus2 when a new interface is opened. The objective was to see where users have struggles. Identifying areas of struggle could have helped improve the usability of the software (e.g., by implementing a wizard functionality, making help-tours or similar).

3.1.3 Analytics

Resulting from the data collection process, the researchers performed some analytics on the data collected from Kactus2. Statistics on times between two actions, per session, were shown on a table (cf. Table 3.1). Another table (cf. Table 3.2) shows the statistics of the toolbar button presses.

Two visualizations were also created at that time. The first one (cf. Figure 3.3) is a simple bar graph, that plots the time needed for a user to unlock a document after opening it. This represents the fourth statistics in the table 3.1. The second one (cf. Figure 3.2) is an event path graph where each state represents a unique action. The size of an arrow between two states $s1$ and $s2$ represents the probability that in the logs, *action 2* is recorded directly after *action 1*. No analytics were made on the data collected from the opening of help pages.

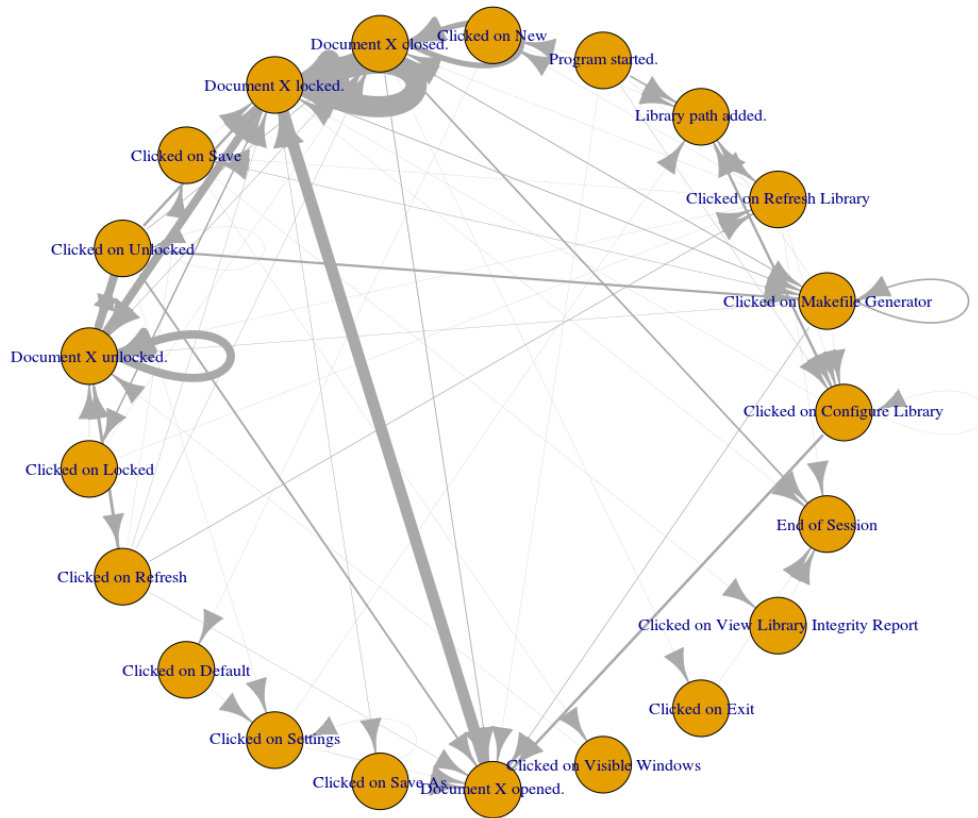


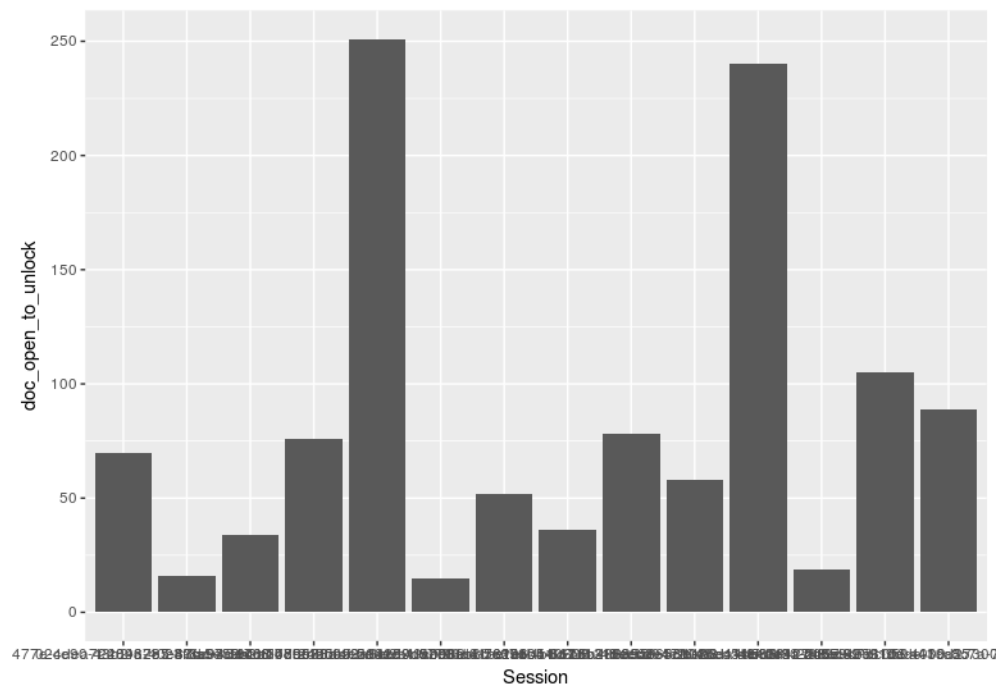
Figure 3.2. Event path graph.

Table 3.1. Session times between two actions (in seconds).

From: To:	Start Library	Library Doc Open	Start Doc Open	Doc Open Unlock	Start Unlock
1	160	347	507	70	577
2	174	176	350	16	366
3	259	247	12	34	46
4	110	28	138	76	214
5	22	68	90	251	341
6	25	474	499	15	514
7	26	80	106	52	158
8	96	283	379	36	415
9	56	24	80	78	158
10	83	146	229	58	287
11	62	965	1027	240	1267
12	418	44	462	19	481
13	19	419	439	105	543
14	8	41	49	89	138

Table 3.2. Statistics on toolbar button presses.

Clicked on...	Count	Clicked on...	Count
About	1	Save	27
Configure Library	35	Save All	4
Default	1	Save As	6
Exit	1	Select Tool	3
Help	1	Settings	4
Interconnection Tool	12	Undo	3
Locked	12	Unlocked	165
Makefile Generator	116	View Library Integrity Report	1
MCAPI Code Generator	1	Visible Windows	2
Refresh	21	New	2
Refresh Library	11		

**Figure 3.3.** Time for document opened to unlocked, per session (in seconds).

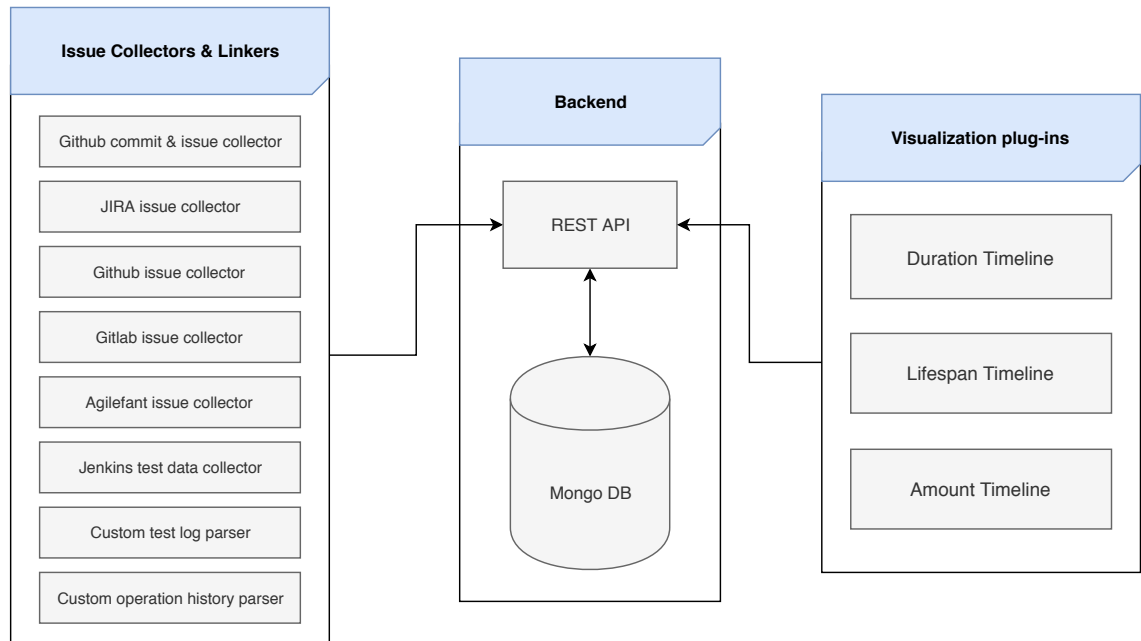


Figure 3.4. Implementation of the framework, from [36] (updated). Issue Collectors and Linkers are examples of usages of this framework.

3.2 Technical Framework

3.2.1 Database

The logs taken from Kactus 2 are parsed and formatted into the Unified Model for Software Engineering Data (UMSED) presented in 2.5. To use this data model, a framework is presented for sending, storing, and retrieving the data [36]. This framework uses a NodeJS⁵ server running a MongoDB⁶ database. To send and retrieve data, a REST API is used.

For each data source, a pair of collector and linker must be implemented. The collector retrieves the data from the original source and converts it into the data model. The linker is used to send the data and to create the relations between events and constructs in the database. The visualization then takes care of using the data for creating visualizations (cf. next subsection). Figure 3.4 shows the implementation of this framework.

3.2.2 Visualization Plugin

In [36] is also presented a visualization plugin, which uses the D3⁷ framework. It is an open-source JavaScript library for manipulating documents based on data. D3 (for Data-Driven Documents) uses HTML5, SVG, and CSS standards for producing dynamic,

⁵NodeJS - <http://www.nodejs.org>

⁶MongoDB - <http://www.mongodb.org>

⁷D3.js - <https://d3js.org>

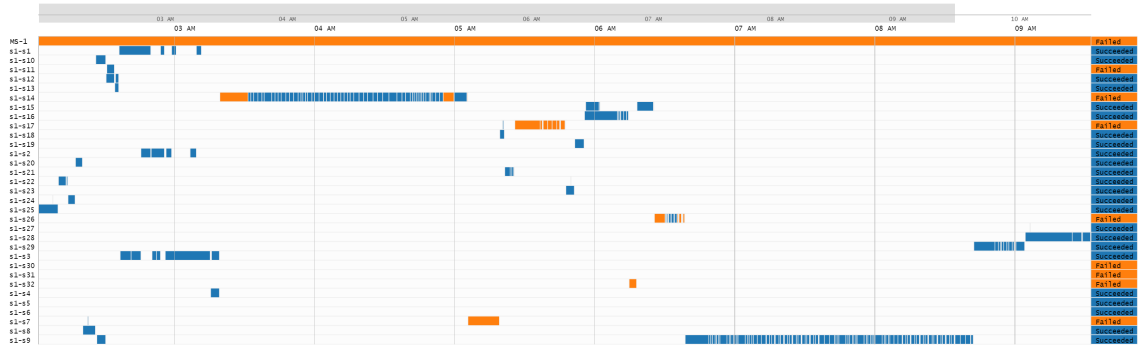


Figure 3.5. Visualization of test log information, from [36].

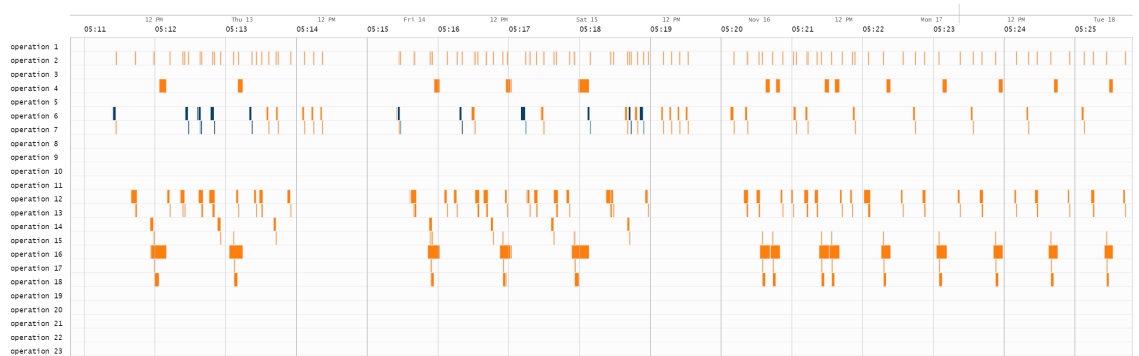


Figure 3.6. Visualization of usage logs, from [36].

interactive data visualizations in web browsers. The plugin retrieves the data from the database, converts it into the visualization data format and generates the visualization based on the required template.

The timeline-based template proposed is parameterized. Users can select which attribute of which artifact will be used as the y-axis. This allows very precise grouping of events, as events related to the same y-axis identifier will be drawn on the same line. Figures 3.5, 3.6, and 3.7 show examples of different uses of this template.

3.2.3 Applications of the Framework and Visualization Plugin

In [36] are mentioned several parsers as a proof-of-concept for the framework. These are parsers for Jira, Github, test log, and product usage log data. The data is parsed and sent to the database, and accessed through the REST API. For the visualization plugin, [36] gives two visualization examples, for usage logs (figure 3.5) and test logs (figure 3.6). Each line of the visualization represents a constructs: *test sets* in figure 3.5 and *operation types* in figure 3.5. In both visualizations, the length of the bars on each line represent the duration of the execution (respectively of the test and of the operation). Colours are used to distinguish between failed and successful operations.

Hylli et al. present a tool for collecting issue management data from different services [21]. It describes an intermediate model for data from a specific domain, that is then



Figure 3.7. Visualization of issue management data, from [21].

converted in the UMSED. The visualization plugin is then used to display this data. Figure 3.7 shows how the visualization plugin has been used to visualize issue lifespans and events from a Github project. Each line represents an issue, the length of the bar being its duration. Circles on the lines show events related to the issue.

Patrushev uses a modified version of the UMSED [46]. It is used to handle software data form Github repositories, but the data is not intended to be used by the visualization plugins. A different framework is thus implemented to take care of this modified data format. The data is here destined to be used for training of a self-organizing map, in order to analyze software project management patterns.

4 METHODOLOGY

This chapter describes the methodology and process of constructing this thesis. Section 4.1 shows how the Unified Data Model presented in section 2.5 was implemented. Section 4.2 describes the process of designing and developing visualizations for the data of Kactus2. Section 4.3 details how the visualizations were evaluated.

4.1 Data Model

The first step towards the objectives (cf. Section 1.1) was to insert the data coming from Kactus2 (described in section 3.1) into the UMSED presented in section 2.5. The data model distinguishes primarily *events* and *constructs*, making them the main building blocks of the model. It was thus necessary to start by analysing the logs and identifying what would qualify as an *event* or as a *construct*.

These are examples of how log entries are formatted.

```
25.01.2017 08:32:24 user A in session B: Document C unlocked.  
27.01.2017 09:34:12 user P in session Q: Help page R opened.  
29.01.2017 10:01:52 user X in session Y: Clicked on Z.
```

Listing 4.1. Example of 3 log entries.

4.1.1 Elements of the Data Model

From the definition of a **construct** presented in the data model - an aspect of software engineering that is interesting for visualization and analysis purposes, the objects on which events are performed - four elements stand out:

- **User** The user is the starting point of the logs. He is the one performing actions on the other constructs.
- **Session** Sessions are created by users, and all actions on documents and help pages are contained within a session.
- **Document** Documents can be opened, closed, locked and unlocked by a user within a session.

- **Help page** Help pages are opened by a user within a session.

These four software engineering concepts are interesting for analysis, and provide essential information on the usage of the software. The third line in the log example shows another SE concept: what the user has clicked on. This was not considered to be relevant as a construct (but is not ignored none the less, as explained below).

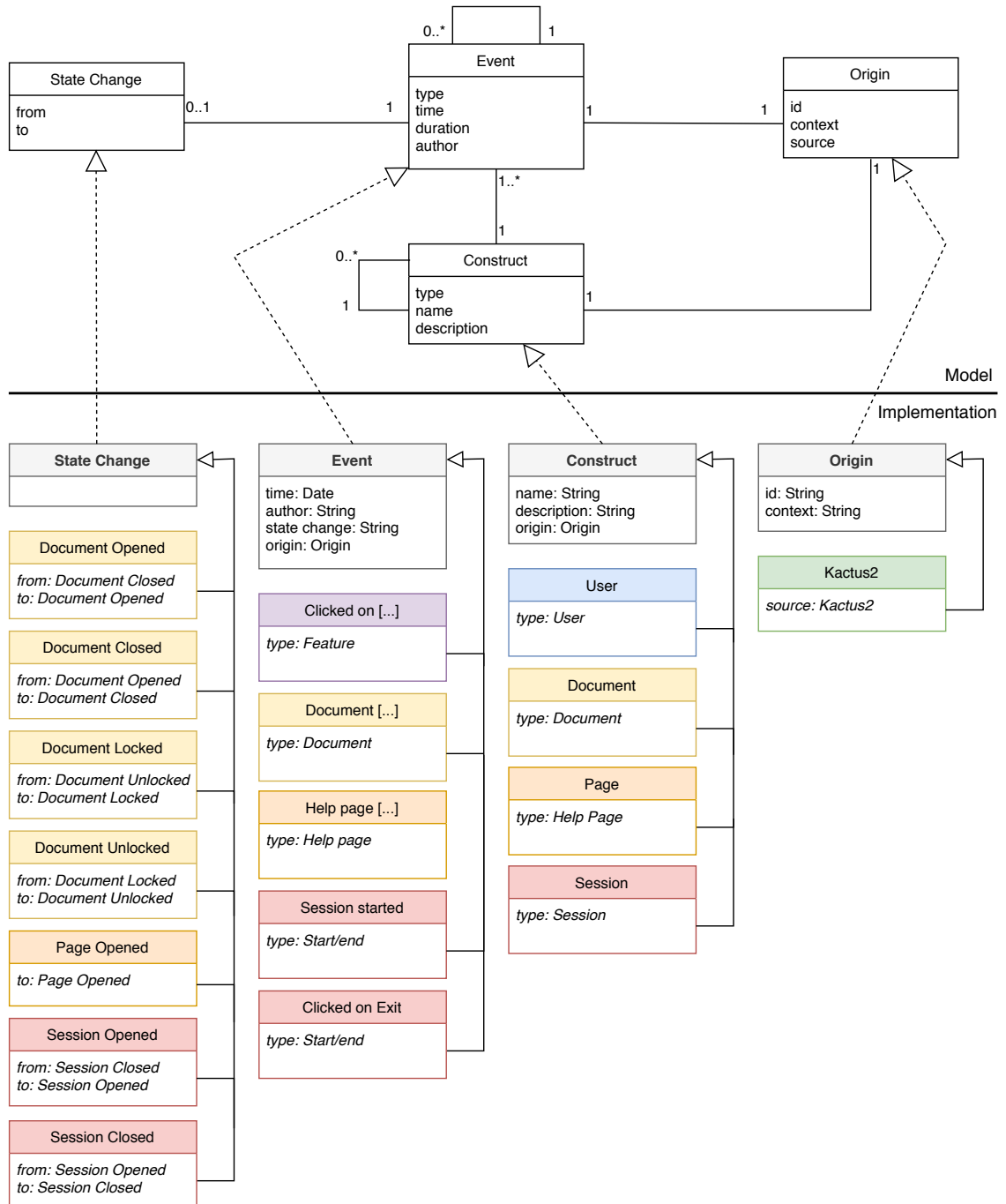


Figure 4.1. Implementation of the Unified Data Model for Software Engineering Data. Yellow elements are related to documents, orange to help pages, red to sessions. Purple are features, blue are users, green is the origin.

Once constructs were identified, we looked at what **events** (i.e., actions performed by a

user on one or more construct(s) at a certain time) appear in the logs. The events were separated into four categories, which result from how the data was collected (cf. Section 3.1.2 and the three questions that lead the data collection process):

1. **Start/end** The events marking the beginning of a new session, and its end.
2. **Documents** The events indicating that a document has been *opened*, *closed*, *locked* or *unlocked*.
3. **Help pages** The events indicating that a help page has been *opened*.
4. **Features** The events indicating that a *feature* has been used (i.e., the button presses on the toolbar of Kactus2).

These four types of events encompass all the actions recorded in the logs. The events were marked as **state changes** (i.e., optional attribute of an event used when the event triggers a change of state in the construct it is linked to) when adequate. This was the case for the first three types of events enumerated above. Finally, the **origin** (i.e., source and context of the data) was obvious. The source is the log file used, and the context is the software itself: Kactus2.

Figure 4.1 shows the implementation of the Unified Data Model for Software Engineering Data.

4.1.2 Links Between the Elements of the Data Model

In the model shown in 2.17, multiple events can be linked to one-another. This was not necessary in our case. But we do link several constructs together, and of course constructs to events. One *user* is linked to one or more sessions. A *session* is comprised of one or more events. An *event* can be a *state change*, and can be linked to at most one help page or one document. *Help pages* and *documents* are linked to the user that uses them and the session in which they exist. In this configuration, a document or help page is never accessed by different users.

4.1.3 Data Parsing and Linking

The integration of the logs into the database was designed to remain coherent with the previous implementations of the framework (cf. Figure 3.4, *Collectors and Linkers*). Three modules take care of handling the data upstream:

1. The **Collector** receives input from the user (e.g., which log file to use), and lets the *Parser* collect the data, then send this data to the *Linker*.
2. The **Parser** parses the desired log file and returns the list of events, state changes, users, sessions, documents and help pages.

- The **Linker** formats the received data into the *events* and *constructs* as defined by the UMSED implementation (cf. Figure 4.1). It then sends these to the database. Finally, it creates the links between the elements of the database.

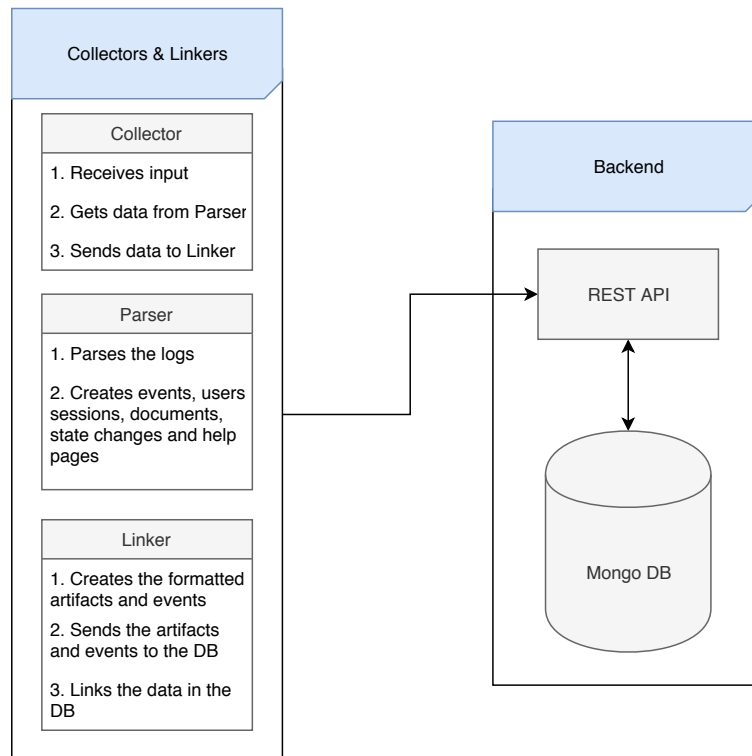


Figure 4.2. Implementation of the integration of the data in the database.

4.2 Visualizations

4.2.1 Session Timelines

The first visualization - named *Session Timelines* - was designed to display the lifespans of the sessions, and the events that occurred during these lifespans. The objective was to be able to compare easily the evolution of the sessions between each other. The starting point for this visualization was the *Issue Timeline* visualization shown in figure 3.7 which displayed the lifespan of issues from Github.

The *Session Timelines* visualization can be seen on figure 4.3. The visualization should be read line-by-line. Each line represents a session of a user. The user is indicated (in color) on the right of each line.

All sessions start *artificially* at the same time to display the events in parallel. This allows to compare all sessions at the same time, independently of the time and date of their creation.

The events recorded are displayed on the session timelines by colored circles. Four types

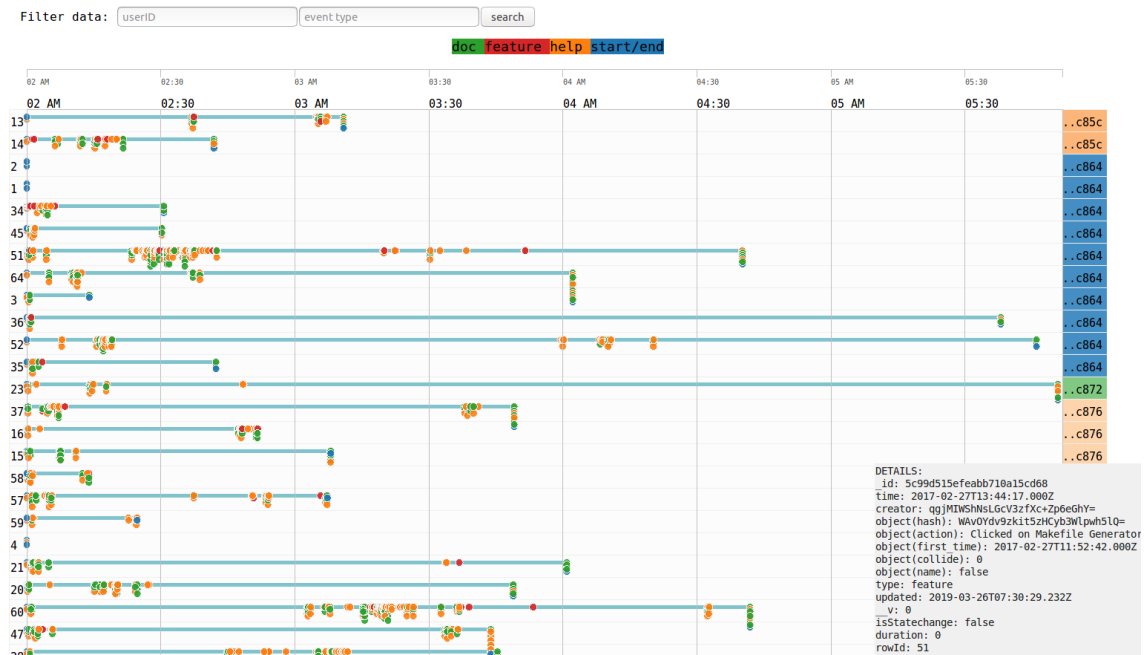


Figure 4.3. Session Timelines visualization.

of events are distinguished:

1. **Start/End** in blue. These events indicate the opening and closing of the session.
2. **Documents** in green. These events indicate actions performed on documents by the user during the session (*document open, locked, unlocked, or closed*).
3. **Help pages** in orange. The events indicate the opening of a help page in the software.
4. **Features** in red. These are the events recorded by the clicks of the user on the main task bar (toolbar button presses).

When hovering over an event circle, the bottom-right tool-tip box provides more details on the event (action recorded, page/document related, etc).

On the top, the *Filter data* option allows to filter by user and/or event type. The coloured legend indicates the type of the events displayed on the session lifespan.

Between the legend and the actual visualization, the interactive time-selector allows to zoom-in the visualization to get a more localized view of the events.

4.2.2 User Timeframe

The second visualization - named *User Timeframe* - was designed to show in more detail the unfolding of events in each session individually. The main objective was to see the flow of actions in a session, through the documents opened and help pages viewed¹.

¹The opening of help pages is recorded automatically each time the user enters a new view in the software. Therefore tracking the help pages indicates us what the user is viewing at each moment.

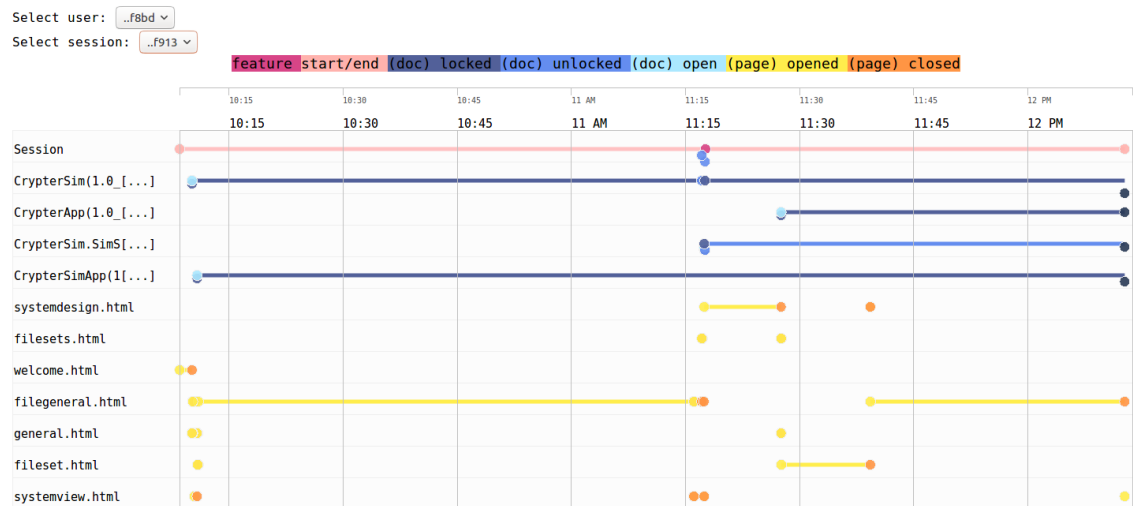


Figure 4.4. User Timeframe visualization.

The visualization is divided in three parts (cf. Figure 4.4):

1. **Session** (represented in pink). The first line displays the lifespan of the whole session. On this lifespan the events classified as *features* are represented by darker pink circles. These are the events recorded by the clicks of the user on the main task bar.
2. **Documents** (represented in blue). The next lines (three lines in the example) show the lifespans of the documents opened by the user during the session. Different shades of blue indicate the state of the document (*open*, *locked*, or *unlocked*). The events that lead to the changes of the state are indicated by circles of the same color.
3. **Pages** (represented in yellow). The last lines (six in the example) show the lifespans of help pages viewed by the user. In essence, it shows the path taken by the user through the session in the software. Yellow circles indicate the opening of a page, orange circles indicate the switching to another page.

Hovering over the lifespans or events will display the tool-tip box (as in the first visualization), giving more indications. The names of the documents and help pages and displayed left of the corresponding lines. Hovering over the names will display the complete name in the tool-tip box.

On the top left corner, we are able to select the user and session that we want to visualize. The legend and time-selector work in the same way as the first visualization.

4.3 Evaluation

We will use two complementary approaches (semantic and syntactic analysis) for evaluating the two visualizations developed. An ontological evaluation of the visualizations

will be made to provide a semantic analysis. The syntax of the visualizations will be evaluated in regards to the *Physics of Notations* [41]. Looking at both the syntax and semantics of the visualizations will allow us to assess both *what* is represented and *how* it is represented. Our hypothesis is that the ontological evaluation will be likely to find flaws or incompleteness in the data model used. Because *what* we can represent is inherently limited by the design of the data model itself. On the other hand, the syntactic analysis should give us insight into the adequacy of the visualization templates for representing the data: *how* the data is represented depends on the visual notations developed.

4.3.1 Semantic Analysis

Similarly to Opdahl and Henderson-Sellers in their ontological evaluation of the UML, we will base our semantic analysis on the Bunge-Wand-Weber (BWW) model of information system [44]. The BWW model (cf. [61, 62]) is derived from Mario Bunge's ontological model, which is inspired by systems theory [7, 8].

The semantic domain was defined by analysing the data model (UMSED) discussed in section 2.5, which was used to model the data collected. Hence our semantic domain is the set of semantic constructs that are described by or can be inferred from the data model.

The detailed tables can be found in the following appendices:

1. **BWW-model:** Appendix B. The set and description of the ontological concepts.
2. **Semantic domain:** Appendix C. The set and description of the semantic constructs.
3. **Representation mapping:** Appendix D. The mapping between the ontological concepts and the semantic constructs.
4. **Interpretation mapping:** Appendix E. The mapping between the semantic constructs and the ontological concepts.

The objective of the representation mapping from the BWW-model to the semantic domain is to:

1. Identify *redundant constructs*, which overlap semantically with others.
2. Identify *construct deficits* in the semantic domain.

The interpretation mapping from the semantic domain to the BWW-model is needed to:

1. Identify constructs that are not problem-domain oriented: *construct excess*.
2. Identify *overloaded constructs*, that are intended to represent different kinds of phenomena.
3. Define precisely the *meaning* of each semantic construct.

Our assumption is that this semantic analysis will reveal potential weaknesses in either the data collection process, or the data model used to contain this data.

4.3.2 Syntactic Analysis

To complete the evaluation of the visualizations, we will confront our visual notations with the *Physics of Notations* [41]. Described in section 2.4.1, this theory gives rational, evidence-based principles for designing effective visual notations. This approach is complementary to the ontological analysis, because such an evaluation will not differentiate two visual notations that have the same semantics but different syntax. This is critical as we know that *form* has a comparatively greater effect on cognitive effectiveness than *content* in visual representations in information systems [26, 52]. This theory has been used for example, to evaluate the cognitive effectiveness of the BPMN 2.0² visual notation [18]. Our visualizations will be evaluated on the nine principles of PoN, considering as well the interactions between the principles.

Questionnaire - Session Timeframe visualization	
Question	Answer type
1 How easy was it to understand the visualization?	Scale 1 to 5 with 1 = Very hard, 5 = Very easy
2 Could the visualization be used to assess the usage of Kactus2?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
3 What is in your opinion the most important piece(s) of information that can be viewed with the visualization?	Open answer
4 Could the visualization be used to improve the software in any way?	Yes/No
5 Would you find personal use in this visualization as it is?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
6 What if the data had been collected on "real users" (as opposed to lab students)?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
7 Rate the usefulness of: the user filter	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
8 Rate the usefulness of: the event type filter	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
9 Rate the usefulness of: the event type legend	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
10 Rate the usefulness of: the time-selector	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
11 Rate the usefulness of: the tool-tip	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
12 How would you improve this visualization?	Open answer

Table 4.1. Questions on the Session Timeframe visualization. Questions were asked in this exact order.

4.3.3 Questionnaire

A third method to evaluate the visualizations was to gather feedback from potential users (i.e., developers of the software Kactus2). This feedback took the form of a questionnaire that was sent to two developers. The questionnaire was divided in two parts, one for each visualization. Each part started by a description of the visualization, aided with

²Business Process Modeling Notation - <http://www.bpmn.org/>

screenshots, and was followed by a series of questions. As the visualizations are interactive, explaining it only with screenshots without the ability for the developers to try it out themselves is not optimal. The rationale for this decision was the convenience of the procedure, as the platform which hosts the visualizations and technical framework is unstable and hardly portable. Asking the developers to try to install it themselves and load the data might have discouraged the respondents. Performing a face-to-face interview and providing the up-and-running visualizations was excluded as well, as it was also deemed inconvenient for the developers.

The objective of the questionnaire was to have an idea of the actual usefulness of the visualizations, as well as the potential ease of use and understanding. The questions asked for each visualization are listed in tables 4.1 and 4.2. The questions were asked in the order shown in the tables, after the description of each visualization. Questions 1, 2 and 4 were followed by an optional comment field. We mentioned that questions should be left blank if the respondent did not know the answer or simply did not wish to respond. Before starting the questionnaire, we asked for consent to use the answers in the context of this thesis. The questionnaire was anonymous, in the sense that we do not know which developer is behind each response (i.e., names were not asked in the questionnaire).

Questionnaire - User Timeline visualization		
Question		Answer type
1	How easy was it to understand the visualization?	Scale 1 to 5 with 1 = Very hard, 5 = Very easy
2	Could the visualization be used to assess the usage of Kactus2?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
3	What is in your opinion the most important piece(s) of information that can be viewed with the visualization?	Open answer
4	Could the visualization be used to improve the software in any way?	Yes/No
5	Would you find personal use in this visualization as it is?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
6	What if the data had been collected on "real users" (as opposed to lab students)?	Scale 1 to 5 with 1 = Not at all, 5 = Absolutely
7	Rate the usefulness of: the legend	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
8	Rate the usefulness of: the time-selector	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
9	Rate the usefulness of: the tool-tip	Scale 1 to 5 with 1 = Not useful at all, 5 = Very useful
10	How would you improve this visualization?	Open answer

Table 4.2. Questions on the User Timeline visualization. Questions were asked in this exact order.

We tried to follow good practice of questionnaire design. But we are not experts in this discipline and bias is ultimately unavoidable. In general, we aimed to:

- Go from general questions to specific questions, to avoid the framing effect.
- Avoiding *double questions* (i.e., questions that contain a non-trivial assumption).
- Avoiding phrasing questions using double-negations.
- Allowing the respondent to abstain from answering.
- Restrain from giving examples of answers in the question.

Furthermore, for questions with *scaled answers*, we chose an odd scale (in our case, 1 to 5). An odd scale allows the respondent to select the *middle* option in the scale, which is the easy answer if the respondent is unsure. While an even scale (e.g., 0 to 5) does not permit this: the answer will be situated either on the left or the right of the middle. This kind of scale might seem like forcing the hand of the respondent to not be neutral.

5 RESULTS

This chapter will present the concrete implementation of the visualization, the results of the semantic (ontological) and syntactic analysis, as well the responses to the questionnaire discussed in section 4.3.

5.1 Implementation

This section will detail the technical implementation of the visualizations, changes and additions to the pre-existing code base of the framework and templates. The original source code can be found on Bitbucket¹. The forked version that we are working on can be found on Github².

5.1.1 Technical Framework

The part of the code base that is responsible for the handling of the data is located in `datasources`. The structure of that part can be seen in figure 5.1. Each sub-folder (such as `agilefantparser`) is responsible for fetching the data from the source and sending it to the database. As we used a new data source, we created a new sub-folder `log` to handle this.

We followed the same structure as the previous data handlers by using a three-file system composed of three main files.

- `collector.js` receives the input (log file name) from the user and sends the data from the parser to the poster.
- `getdata.js` parses the log file given as input and returns a coherent data structure.
- `poster.js` transforms the data received into the data format specified by the database (constructs, events and links), and sends this data to the database.

¹<https://bitbucket.org/rimina/n4s-visu/src/master/>

²<https://github.com/coin-quin/vis-a-vis>

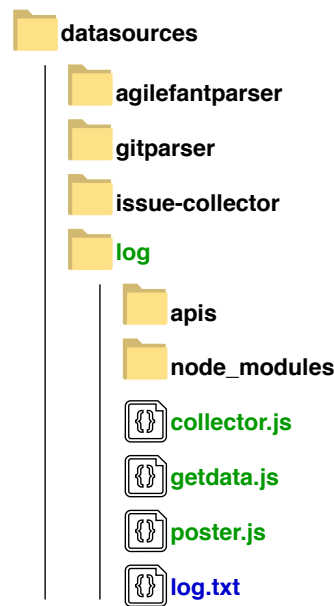


Figure 5.1. Source code structure: data sources. In green are the new folders and files created. In blue is the data source.

5.1.2 Visualizations

The code base of the visualizations is slightly more complex, and can be seen in figure 5.3. We will detail only the new files created (in green and purple). The modified files (in orange) were only slightly altered, mainly to accommodate for the new files. Again, we followed the same file organization as the previous visualizations did.

Views `session_timeframe.html` and `user_timeline.html` are the HTML pages which are the first user interfaces (UIs) of the visualizations. On them are displayed the filtering options and the data query specifiers. Figure 5.2 shows the interface of the *User Timeline* visualization.

UI Utilities The files `x_query_ui.js`³ retrieve the contents from the aforementioned views. The files `x_template.js`³ define the *containers* and *svg elements* that will be used to frame the diagram.

Visualizations The files `x_main.js`³ are the main files of the visualizations. They initialize the diagrams and data structures, and load the data onto the graphical elements.

Visualization Templates The files `session_timeframe.js` and `user_timeline.js` define the graphical characteristics of the diagrams used in the visualizations. The file

³Replace x with either `user_timeline` or `session_timeframe`.

Data Filtering

Time range filtering

Leave the fields empty if you don't want to filter data based on time range.

start date: start time (hh:mm):
 end date: end time (hh:mm):

Data origin filtering

Source file:

User filter

User name:

Figure 5.2. Starting UI of the User Timeline visualization.

`data_selector.js` is used by the *User Timeline* visualization. It is the user- and session-selection component used to browse between the various users and sessions.

Data processors The files `x_dataprocessor.js` are used to perform various operations on the data used by the visualizations. Such operations include parsing, sorting and filtering for example.

5.2 Ontological Analysis

The ontological analysis will look at the mapping between the ontological concepts of the Bunge-Wand-Weber model [61, 62] and the semantic constructs. The detailed tables can be found in the following appendices:

1. **BWW-model:** Appendix B. The set and description of the ontological concepts.
2. **Semantic domain:** Appendix C. The set and description of the semantic constructs.
3. **Representation mapping:** Appendix D. The mapping between the ontological concepts and the semantic constructs.
4. **Interpretation mapping:** Appendix E. The mapping between the semantic constructs and the ontological concepts.

We used the representation mapping to highlight potential *construct deficits* and groups of *redundant constructs* in the semantic domain. The interpretation was used to reveal *overloaded constructs* and *excessive constructs* in the semantic domain.

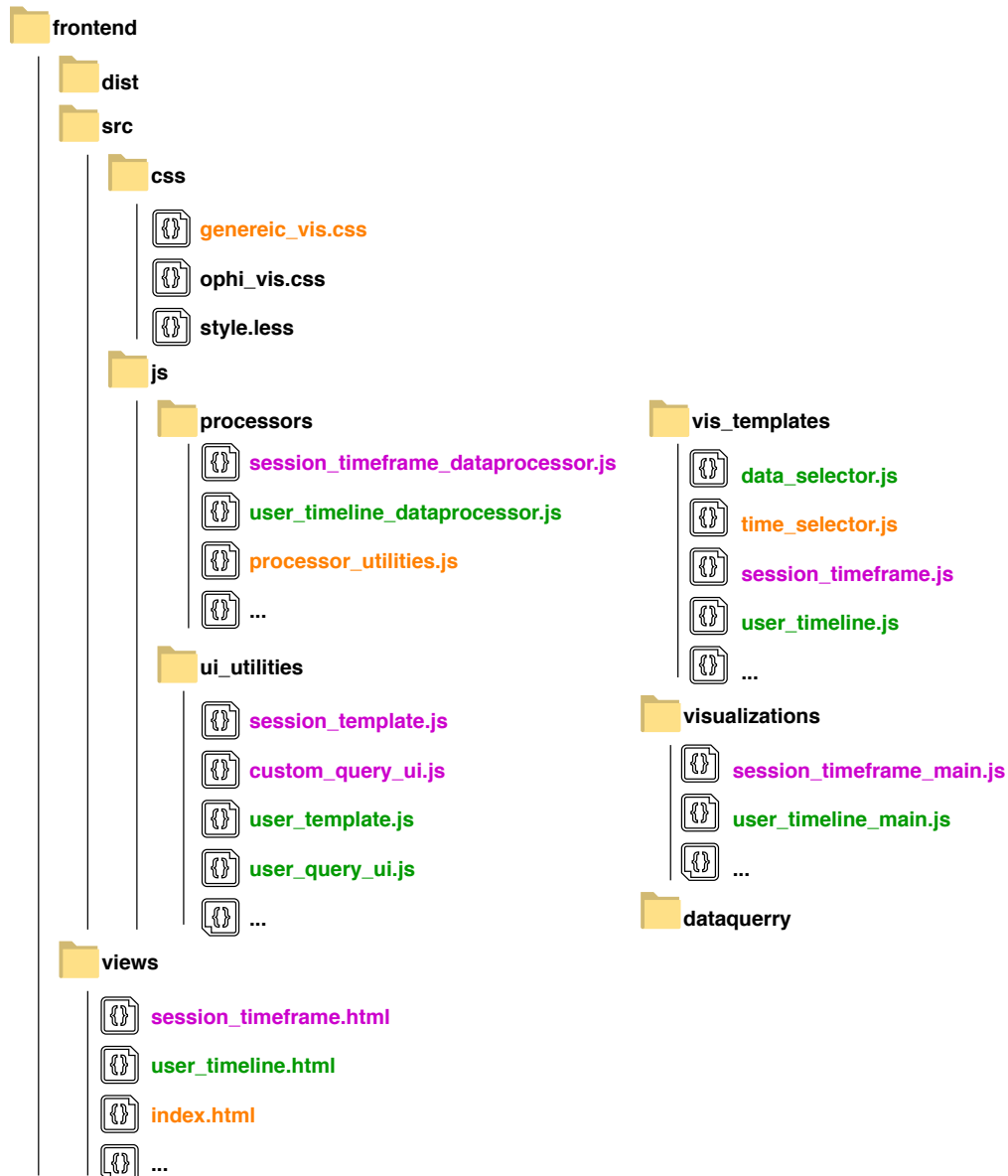


Figure 5.3. Source code structure: frontend. In green are the new files created for the User Timelines visualization. In purple are the new files created for the Session Timeframes visualization. In orange are the existing files modified. Three dots indicate files of other visualizations that we are thus not using.

5.2.1 Redundant Constructs

Construct redundancy occurs when two or more semantic constructs represent the same ontological concept. We must distinguish two types of redundancies in our semantic domain:

1. *Technically redundant* constructs represent the same BWW-concept because they are sub-types of a more general (non-redundant) construct. This is not particularly problematic.
2. *Genuinely redundant* constructs actually represent the same BWW-concept in the

Sub-types of SEM-entities	SEM-object, SEM-user, SEM-session, SEM-document, SEM-help page, SEM-action, SEM-event, SEM-state change.
Sub-types of SEM-entity properties	SEM-object property, SEM-user property, SEM-session property, SEM-document property, SEM-help page property, SEM-action property, SEM-event property, SEM-state change property.

Table 5.1. *Technically redundant semantic constructs.*

problem-domain. This potentially indicates a weakness in the semantic domain.

Semantic constructs that represent sub-types of BWW concepts We have identified three main cases where semantic constructs are technically redundant because they are sub-types of other constructs:

1. **BWW-thing** is represented by an instance of a SEM-entity, as well as all its subtypes.
2. **BWW-property** is represented by SEM-entity property, as well as all its subtypes.
3. **BWW-class** is represented by SEM-object and SEM-action, as well as their subtypes.

The types and subtypes are shown in table 5.1.

Genuinely redundant semantic constructs We have not found genuinely redundant constructs. This may be explained by the fact that the semantic domain of our problem is quite simple and restricted already.

5.2.2 Construct Deficit

Construct deficit occurs when a BWW-concept is not represented by any SEM-construct. We detail the major deficits here:

- The **BWW-property function** of a thing maps the thing to some value. It represents how the property of a thing changes over time. The only changing property of the entities of our semantic domain is the *state* of a SEM-object, which is a BWW-intrinsic property. What represents this function is thus the *lifespan* of the SEM-object. But the lifespan is not in itself a semantic construct of our model.
- A **BWW-law property** (natural- or human-law) of a thing is a rule that restricts the properties of a thing. While rules are indeed given both by the data model (e.g., a SEM-state change is linked to exactly one SEM-event) and the data source itself (e.g., a SEM-action is linked to exactly one SEM-session), there are no semantic constructs that represent these laws.

- A **BWW-process** in a thing is a chain (or tree) of events on a thing, or states of that thing. The state of a SEM-object is (as previously mentioned) an intrinsic property of that object. If the list of SEM-events linked to a SEM-object is indeed a construct, that the list of states is not. The BWW-process is thus only partially represented.
- The **BWW-history** of a thing is the chronologically ordered list of states that a thing traverses in time. For the same reason hereabove stated, the history of a SEM-object is not represented by a semantic construct.
- **BWW-composite and component** things (and the related notions) have no representation in our semantic model. All SEM-entities exist by themselves, even though they may be structurally linked to others.
- The **BWW-system environment** is the set of things that interact with the system but are not part of it. Our model has not semantic constructs for representing this.

5.2.3 Construct Overload

Construct overload occurs when a semantic construct can be used to represent *several* different BWW-concepts. Again, this is only problematic if the different BWW-concepts are not sub-types of each other. The main construct overload identified comes from **SEM-object property** and is problematic. This can be interpreted as any BWW-property of an SEM-object such as *name* or *state*. But the *state* of a SEM-object can also be interpreted as a BWW-state (stable or unstable) of a SEM-object.

5.2.4 Construct Excess

Construct excess occurs when a semantic construct does not represent *any* BWW-concept. There are two possible types of excess. Firstly, *genuinely excessive* semantic constructs, which even though they are intended to represent the problem have no BWW interpretation. And secondly, the semantic constructs that are *not problem-oriented*, i.e., they are useful parts of the model for development or comprehension-related purposes but are not meant to represent the problem itself. We have not identified any construct excess in the semantic domain of our problem.

5.3 Syntactic Analysis

In the syntactic analysis, we consider at the nine principles for designing cognitively effective visual notations proposed by Moody in the *Physics of Notations* [41]. Both visualizations - **Session Timeframe** and **User Timeline** - will be discussed in terms of their cognitive effectiveness.

The Physics of Notations looks in detail at the visual syntax of notations: the combination of the **visual vocabulary** (the set of graphical symbols) and **visual grammar** (the set of compositional rules). It was thus necessary to define the **visual syntax** of our visual notations. As our visualizations are quite simple, so are their syntax. The syntax of the *Session Timeframe* visualization can be seen in tables 5.2 (visual vocabulary) and 5.3 (visual grammar). The syntax of the *User Timeline* visualization is slightly more extensive and can be seen in tables 5.4 (visual vocabulary) and 5.5 (visual grammar).






Visual vocabulary			
Graphical symbol	Detail	Image	Meaning
Line	light blue color		lifespan of a session
Circle	green color		document-related event
	red color		feature-related event
	orange color		help page-related event
	blue color		session start/end event
Text	highlighted (right-hand side)		user name

Table 5.2. Visual vocabulary of the *Session Timeframe* visualization.

Compositional rules	
Graphical representation	Meaning
Circle placed along a (light blue) line at position x (seen from timeline)	The event (circle) related to the session (line) occurred at relative time x (of timeline)
Highlighted-text on the right of a line	The session (line) has been created by user y (text)
The (light blue) line starts/end at position x (seen from timeline)	The session (line) starts/end at relative time x (of timeline)
Circles are "piled up" on a line at position x (seen from timeline)	Events (circles) in a same session (line) occur at the same relative time x (of timeline)

Table 5.3. Visual grammar of the *Session Timeframe* visualization.

5.3.1 Semiotic Clarity

Semiotic clarity looks at the mapping between the visual syntax and the semantic constructs. The set of semantic constructs forms the **semantic domain**: the domain that can be visualized. The semantic domain was previously defined in the ontological analysis, and can be seen in the appendix C.

Symbol redundancy There are no synographs in either of the visualizations. Different graphical symbols cannot be used to represent the same semantic construct. Though SEM-events can indeed be represented (in both visualizations) using circles of different color, that color depends on the *type* of the event, hence its instance (i.e., they are not


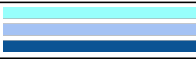






Visual vocabulary			
Graphical symbol	Detail	Image	Meaning
Line	light pink		lifespan of a session
	blue (three shades)		lifespan of a document
	yellow		lifespan of a help page
Circle	light pink		session start/end event
	dark pink		feature-related event
	blue (three shades)		document-related event
	yellow		help page opening event
	orange		help page switching event

Table 5.4. Visual vocabulary of the User Timeline visualization.

Compositional rules	
Graphical representation	Meaning
Light pink circle placed along a light pink line	The start/end event (circle) is related to the session (line)
Dark pink circle placed along a light pink line	The feature event (circle) is related to the session (line)
Blue circle placed along a blue line	The document event (circle) is related to the document (line)
Yellow circle placed along a yellow line	The help page opening event (circle) is related to the help page (line)
Orange circle placed along a yellow line	The help page switching event (circle) is related to the help page (line)
The light pink line starts/end at position x (seen from timeline)	The session (line) starts/end at absolute time x (of timeline)
The blue line starts/end at position x (seen from timeline)	The document (line) is opened/closed at absolute time x (of timeline)
The blue line changes shade at position x (seen from timeline)	The document (line) changes state at absolute time x (of timeline)
The yellow line starts/end at position x (seen from timeline)	The help page (line) is opened/closed at absolute time x (of timeline)
Circles are "piled up" on a blue line at position x (seen from timeline)	Events (circles) in a same document (line) occur at the same relative time x (of timeline)
Session selection	Moving between the sessions of a user
User selection	Moving between the users
Blue (three shades) line in session visualization	The document (line) is related to the session
Yellow line in session visualization	The help page (line) is related to the session

Table 5.5. Visual grammar of the User Timeline visualization.

genuinely redundant). Two circles of different colors could not represent the same event. The same can be said for the lifespan lines of the SEM-objects in the *User Timeline*

visualization.

Symbol overload There are no homographs in either of the visualizations. Each symbol has a unique meaning (i.e., is associated with a single semantic construct). Though one circle could represent an instance of a SEM-event, it cannot represent any type of event (that is determined by the color variable) or any other semantic construct. The same can be said here again for the lifespan lines of the SEM-objects in the *User Timeline* visualization.

Symbol excess There are no excess symbols in either visual notations. All symbols used have a meaning, they are used to represent a semantic construct. The *tool-tip* that is displayed when hovering over an event or lifespan can be seen as a *comment* or *annotation*, but does not consume any visual symbol (i.e., it is not excessive).

Symbol deficit Symbol deficit appears differently in the two visualizations, so they will be detailed separately. In the *Session Timeframe* visualization, the following semantic constructs are not represented by graphical symbols:

- SEM-documents are not represented, hence neither is their relationship with the User and Session semantic constructs.
- SEM-help pages are not represented, hence neither is their relationship with the User and Session semantic constructs.
- The only SEM-state change effectively represented is the opening and closing of a session (seen by the start and end of the session lifespan line). State changes related to documents are not represented.
- The relationship between SEM-event and SEM-state change (SEM-relation between SEM-actions) is not represented as well for the other stage changes.

In the *User Timeline*, the SEM-user is not explicitly represented by a visual symbol. This is unless we consider that the visual notation is comprised of all the individual session visualizations, that are navigated through the user and session lists. With this in mind, we can see that SEM-user is represented by the aggregation of the visualizations of its sessions. Through this lens, the *User Timeline* visualization does not present symbol deficit.

5.3.2 Perceptual Discriminability

To measure the perceptual discriminability of two visual symbols, we look at their visual distance. Visual distance is defined by a) the number of visual variables on which they differ, and b) the size of these differences.

Both visualizations consist of two main types of elements (inherited from the data model): constructs and events. Constructs - sessions, documents and help pages - are represented by colored lines of variable length, while events are represented by colored circles of fixed size. Figure 5.4 details the visual variables used in both visual notations.

Visual variables		
Variable	Usage	Meaning
Shape	line, circle	lines represent constructs (session, document, help page); circles represent events
Color	circle color, line color	circles are of different colors depending on the type of event they represent; lines are of different color depending on the type of construct they represent (and in for documents the state of the construct)
Brightness*	circle color, line color	circles of the same color use different shades to further precise the action of the event; blue lines (for documents) use different shades to precise the state of the construct
Size	line length	the length of the construct line indicates the lifespan of the construct
Horizontal position (x)	line start/end positions, circle position	the starting (resp. ending) position of the construct line indicates the starting (resp. ending) time of the lifespan of that construct; the position of a circle represents the time at which the event occurs
Vertical position (y)	line position, circle position	the vertical position of a line represents the construct to which it belongs; the vertical position of a circle represented that the event is related to that construct
Texture	-	-
Orientation	-	-

Figure 5.4. Visual variables used in the visual notations. Texture and orientation are not used. (*)The brightness is only used in the User Timeline visualization.

The two main types of elements present in the visual notations (constructs and events) differ on shape: lines and circles. This respects quite well the *primacy of shapes*, and is enough to clearly discriminate between what is a construct and what is an event.

In the *Session Timeframe* visualization, events and sessions (the only represented construct) differ on both shape and color, bringing *dual coding*. Instances of sessions are differentiated by their vertical position and are labelled, bringing *textual differentiation*. The sessions are further differentiated in size and ending (horizontal) position, which indicates the lifespan of the session. Events differ on three variables: color (type of the event), vertical position (session to which they are related), and horizontal position (time at which they occur).

In the *User Timeline* visual notation, some circles and lines share the same color. Hence events and constructs only differ on shape. Color is here used to distinguish between the types of constructs. Instances of a same construct type are differentiated by their vertical position and are labelled, bringing *textual differentiation*. Within an instance of a document, its state is distinguished by the shade (brightness) of the line. Events differ on four variables: color (type of the event), brightness (action of the event), vertical position (construct to which they are related), and horizontal position (time at which they occur).

5.3.3 Semantic Transparency

Semantic transparency is about the extent to which the meaning of a graphical symbol can be inferred from its appearance. In the visualizations, the lines and circles are not *semantically immediate*: a novice could not guess that one represents a construct and the other an event solely through their shape. But the other visual variables (such as the position and size) give a cue to their meaning. A line placed along a timed-axis is something that has a life duration. A circle in this context is a clue that it might represent something that is punctual: it exists at a certain instant in time. Hence we argue that lines and circles are *semantically translucent*: they provide a cue to the meaning of the symbols but still require some initial explanation.

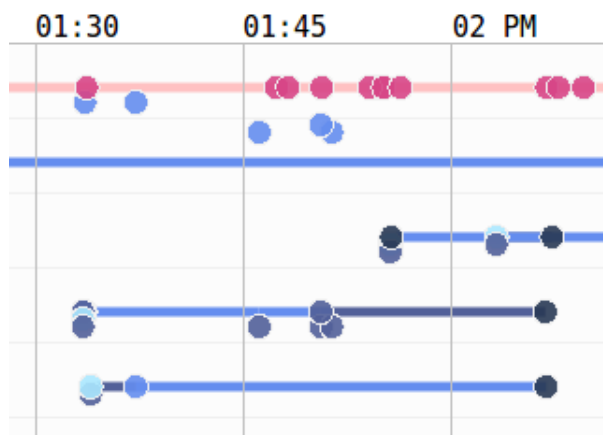


Figure 5.5. *Semantically translucent notation: lines (constructs) and circles (events) placed along a timed-axis.*

A similar argument can be made for the relation between the events and the constructs, as the vertical position of the circles is the one of the lines (events are placed on top of the construct lifespan). In this case, we argue that the relation is close to being *semantically immediate*. Figure 5.5 illustrates the semantic transparency in the *User Timeline* visualization.

5.3.4 Complexity Management

Complexity management is concerned with *diagrammatic complexity*, that is the number of symbol instances on a diagram. Managing this complexity should be done through mechanisms such as *modularization* or *hierarchy*, in order to improve the ability of the notation to represent a large amount of information without causing cognitive overload.

The main way in which our visualizations provide complexity management is through modularization, thanks to the interactive *time selector* that lets the user adjust the time window of the visualization. This allows to zoom-in the horizontal axis, displaying less visual symbols at the same time. Details of the visualizations are that way easier to

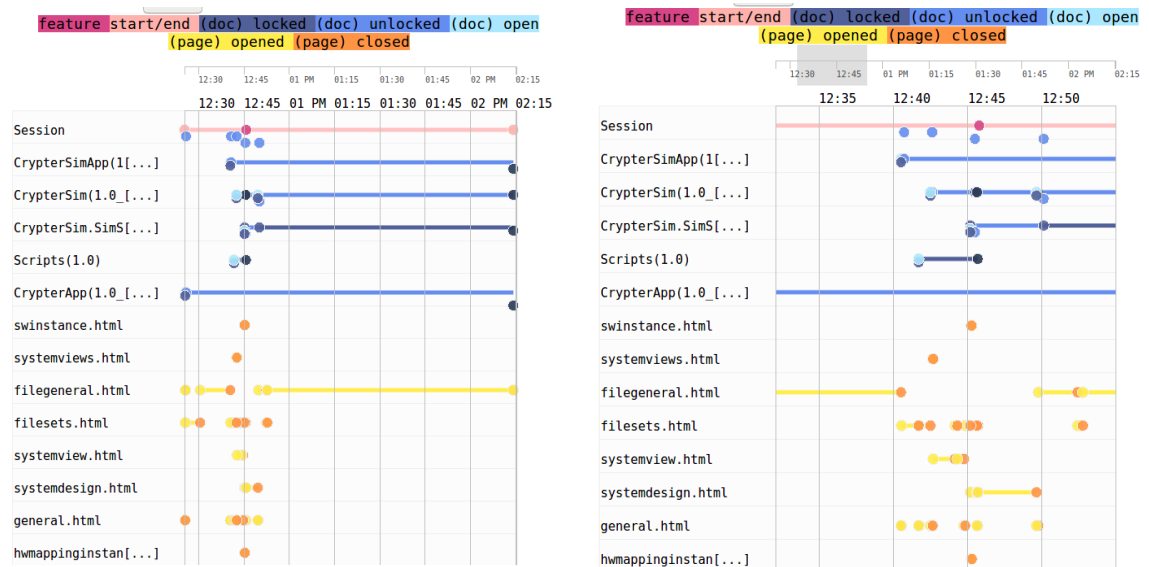


Figure 5.6. Example of modularization with the time-selector in the *User Timeline* visualization. On the left: the default visualization. On the right: the same visualization zoomed-in between 12:35 and 12:55.

notice. Figure 5.6 shows an example of this method on the *User Timeline* visualization.

Modularization is also implemented by the *Session Timeframe* visual notation in a secondary manner by providing a filter option. This allows the user of the visualization to choose which user(s) to display on the diagram, as well as the event type(s). Effectively reducing the number of graphical symbols to process. The *User Timeline* visualization implement hierarchy by dividing the elements shown per session and per user. Hence the graphical elements displayed are limited within a single session of a single user.

5.3.5 Cognitive Integration

Cognitive integration applies when several diagrams are used in conjunction to represent a system. In our case this brings the following question: do we consider that this applies to our two visualizations? As they use the same data source, we can indeed see them as two diagrams that display different parts of the system. As they are different types of diagrams, we are looking at heterogeneous integration. The only way to navigate between those diagrams is through the index page, which contains links to both visualizations. No mechanism is implemented to easily switch between the two. No perceptual or conceptual integration mechanisms are provided to use the visualizations in conjunction with each other.

We then look at the individual visualizations by themselves. As the *Session Timeframe* is a monolithic visualization composed of a single diagram, cognitive integration does not apply to it. More relevant in this regard is the *User Timeline* visualization which, as previously discussed, can be seen as an aggregation of multiple single-session diagrams.

The different diagrams are browsed through the user and session selection options. This way of selecting users and sessions is a mechanism that supports *perceptual integration*: cues that simplify the navigation and transition between diagrams. What it does not support well is *conceptual integration*: helping the user to assemble the information from the multiple diagrams.

5.3.6 Visual Expressiveness

Visual expressiveness looks at the saturation of the graphic design space in a visual notation. This is measured by the number of variables used to encode information, and is the inverse of visual freedom: the number of free (not formally used) variables.

From figure 5.4, we see that most of the visual variables are formally used in both notations. The *Session Timeframe* visualization makes use of five information-carrying variables: shape, color, size, and horizontal and vertical position. The degree of visual freedom is thus of three (only brightness, texture and orientation are not used). The *User Timeline* visualization achieves a visual expressiveness of six by adding brightness to the set of information-carrying variables used by the first visualization.

Color is used to distinguish between event types, as well as between construct types. While this makes use of one of the most cognitively effective visual variables, it is the sole way of discriminating between these types. This is not recommended in a visual notation as color is sensitive to variation in both human perception and display characteristics. No mechanism is implemented to accommodate the use of the visualizations by color-blind people, or to allow the printing of diagram in black-and-white color.

Figure 5.4 shows that while many visual variables are used in the notations, some only use a very limited range of value. This is most noticeable with shape (lines and circles) and color (three to four colors). On the other hand, the position (horizontal and vertical) variables are used to their full range potential.

It is worth noting that throughout this syntactic analysis, we have considered *shades* of colors in the *User Timeline* notation as different levels of brightness of these colors. We do not believe that considering shades as distinct colors (thus not using the *brightness* visual variable) changes the analysis significantly.

5.3.7 Dual Coding

When used as a form of redundant coding, the addition of text to complement graphics (dual coding) can reinforce and clarify the meaning of visualizations. The tool-tip that is displayed when hovering over an event or a construct is a form of *annotation*. It provides additional information on the visual symbol targeted, directly onto the diagram. The legend that helps the user remember which color represents which type of event or construct

can also be considered as an annotation. The *Session Timeframe* visualization makes use of *hybrid* (graphics+text) symbols to display the user name, as well as the session number next to the session lifespan. The *User Timeline* only uses hybrid symbols in the documents and help pages names (next to the construct lifespan) to discriminate between the instances of these constructs.

5.3.8 Graphic Economy

Graphic economy aims at reducing *graphic complexity*: the size of the visual vocabulary of a notation. The objective is to stay below the limit of six distinct alternatives for a single visual variable. Combining several visual variables increases (almost in an additive manner) this limit. Both visualizations have a very limited visual vocabulary (cf. Figures 5.2 and 5.4). This can be explained for both visualizations by the low semantic complexity of the system: there is simply not a lot to show from the data. This is even more visible with the *Session Timeframe* notation, which has a slight symbol deficit.

5.3.9 Cognitive Fit

The cognitive fit theory suggests that a single visual notation is unlikely to be fitted for both *novices* and *experts*, as well as for both *hand-drawn diagrams* and *computer drawing tools*. Hence several visual dialects (e.g., a "pro" and a "lite" one) should be designed within visual notations. Both visualizations are intended only for a single representational medium: a computer screen. Furthermore, they are drawn automatically from the data fed, using the visualization templates developed. In this situation, considering other representational media is irrelevant. Similarly, there is no alternative dialect for communication with experts or novices.

5.3.10 Interactions Between the Principles

Knowing of the interactions between the previously discussed nine principles allows us to look at where *trade-offs* were made, and where *synergies* were exploited, intentionally or not. One of the clear interactions has been mentioned already: having *construct deficit* (from the semiotic clarity principle) in the *Session Timeframe* visualization leads to more graphic economy. For both visualizations, we can also see that having visually expressive notations (good usage of the graphical design space) helps with the perceptual discriminability of symbols. Interestingly, while cognitive integration tends to impact negatively semiotic clarity and graphic economy, this does not seem to be the case in our visualizations.

Questionnaire - Session Timeframe visualization		
Question	Answers	
1	How easy was it to understand the visualization?	Avg: 3.5/5
2	Could the visualization be used to assess the usage of Kactus2?	Avg: 4/5
3	What is in your opinion the most important piece(s) of information that can be viewed with the visualization?	1) The most import in the visualization is the time aspect of the events i.e. what events occur in temporal proximity. 2) The tooltips. Detailed information of the actions could help in developing the user interface to more user friendly. The time it takes for the user to open documentation after opening a document.
4	Could the visualization be used to improve the software in any way?	Yes (100%)
5	Would you find personal use in this visualization as it is?	Avg: 2.5/5
6	What if the data had been collected on "real users" (as opposed to lab students)?	Avg: 4/5
7	Rate the usefulness of: the user filter	Avg: 4.5/5
8	Rate the usefulness of: the event type filter	Avg: 5/5
9	Rate the usefulness of: the event type legend	Avg: 3.5/5
10	Rate the usefulness of: the time-selector	Avg: 4/5
11	Rate the usefulness of: the tool-tip	Avg: 4/5
12	How would you improve this visualization?	1) The tooltip has too much clutter with all the strings of numbers and characters which could be improved by formatting and removing unnecessary data. 2) Change the colours of the orange and red circles. Remove the start / end circles (the line is enough). More detailing on the actions (currently hidden in a tooltip).

Table 5.6. Answers to questions on the Session Timeframe visualization.

5.4 Questionnaire

The two developers responded to the questionnaire. Tables 5.6 and 5.7 list the answers of the questionnaire for the first and second visualizations. The results show that both visualizations were mostly easy to understand once explained. One respondent commented that "The [Session Timeframe] visualization provides a clear layout with distinct colors". For the *User Timeline* visualization, one respondent noted that he "did not understand what the purpose of the session line was". Both visualizations were considered useful for assessing the usage of Kactus2. One respondent noted that "After getting used to it, [the User Timeline visualization] contains detailed information of a single session".

In the *Session Timeframe* visualization, the most important insight provided concerned the time aspect: "temporal proximity" of events and the time needed between the opening of a document and the opening of documentation. Additionally, the extra information contained in the tool-tip was deemed important. In the *User Timeline* visualization, the most important pieces of information were: the user navigation between the documents and help pages, as well as the documents usage.

Both respondents agreed that the *Session Timeframe* visualization could be used to improve the software. One respondent commented that the UI could be improved according to the data, if more details were provided. The other respondent noted that "the visualization could show what features (toolbar buttons etc.) should be grouped together for easy

Questionnaire - User Timeline visualization		
Question	Answers	
1	How easy was it to understand the visualization?	Avg: 3.5/5
2	Could the visualization be used to assess the usage of Kactus2?	Avg: 4/5
3	What is in your opinion the most important piece(s) of information that can be viewed with the visualization?	1) The visualization shows how the user navigates between the different documents/help pages. 2) Time spent and specific actions performed on the different documents.
4	Could the visualization be used to improve the software in any way?	Yes (50%) / No (50%)
5	Would you find personal use in this visualization as it is?	Avg: 3.5/5
6	What if the data had been collected on "real users" (as opposed to lab students)?	Avg: 4/5
7	Rate the usefulness of: the legend	Avg: 4.5/5
8	Rate the usefulness of: the time-selector	Avg: 4.5/5
9	Rate the usefulness of: the tool-tip	Avg: 3.5/5
10	How would you improve this visualization?	1) It might be interesting to see the navigation between documents, so maybe add a new event on the document open-close timeline to show when the user returns to that document. Again, the tooltip could be clearer. 2) Add the possibility to see a sample of all of the sessions, possible filtered by the selected user.

Table 5.7. Answers to questions on the User Timeline visualization.

and fast access". The *User Timeline* visualization was considered as a potential basis for improving Kactus2 by only one (out of two) respondents. He noted that the visualization "displays the need of help files, which could be the result of an unclear user interface".

On average, it seems like the respondents indicated would make little personal use of the *Session Timeframe* visualization. Answers for the *User Timeline* visualization are more positive. For both visualizations, having access to real-use data (as opposed to data collected on students' laboratory sessions) would improve the usefulness of the visualizations.

According to the respondents, the most useful features of the *Session Timeframe* visualization are the *event-type* and *user* filters. The most useful features of the *User Timeline* visualization are the *legend* and *time-selector*.

In the *Session Timeframe* visualization, improvement areas were the *tool-tip*, *color scheme* and *information displayed*. The same comment about the *tool-tip* comes also for the *User Timeline* visualization. Other improvements of this visualization would be *navigation between the documents* and a *grouped view* of all sessions of a user.

6 DISCUSSION

In this thesis, we apply the Unified Model for Software Engineering Data (UMSED) to post-deployment data (PDD), specifically usage data, that had previously been collected from the Kactus2 software. We design two visualizations for this data, based on the technical framework and visualization templates designed for this data model. The objective is two-fold: assessing the adequacy of the UMSED for representing software engineering data from usage logs, and extending the proof-of-concept of the technical framework and visualization templates to visualizations of complex usage data. We find that the data model is adequate for representing accurately the structure of the usage data logs that originate from Kactus2. The results show that it is possible to develop visualizations of software usage data that provide useful insight into the usage of Kactus2, with the aforementioned technical framework.

In this chapter, we will present our interpretation of the results of the ontological and syntactic analysis, as well as the feedback evaluation questionnaire. We will then mention the limitations of our work, and suggest directions for further research.

6.1 Interpretation of the Results

6.1.1 Ontological Analysis

Representation Mapping The semantic analysis of the visualization shows that the model used for the visualizations is **incomplete** with regards to the BWW ontological model. Indeed, several *construct deficits* are apparent, indicating that some BWW-concepts cannot be represented by the semantic domain of our visualizations. In practice, the main deficits are linked to the notion of *lifespan* which is an integral part of the visualization. Yet the lifespan was not mentioned as part of the semantic domain. The reason for it is that the lifespan of a construct represents a complex aggregate of data from different parts of that data model. Furthermore, which elements are part of a lifespan differ between the two visualizations. The other deficits are not to be disregarded nonetheless, as they show potential limitations of the data model.

Interpretation Mapping On the contrary, the results show that the semantic domain of the visualizations is mostly **clear**. The instances of *construct redundancy* are not

problematic, as they are not genuinely redundant. Moreover, no *construct excess* was identified. But a problematic case of *construct overload* was identified. This indicates that the model induces confusion regarding the notion of state and how it should be represented.

6.1.2 Syntactic Analysis

Semiotic Clarity The analysis shows that both visualizations are **clear**. Between the semantic constructs and the graphical symbols, there is no *symbol redundancy*, *excess* or *overload* in the visual syntax of our notations. The instances of *symbol deficits* are not weaknesses either, as they result from a deliberate choice to limit the scope of each visualization to a certain number of elements. Trying to remove deficits would result in an increase of the graphic complexity, which is undesirable.

Perceptual Discriminability We argue that considering the very low amount of graphical symbols in the *Session Timeframe* visualization, the visual distance between the symbols is sufficient. The perceptual discriminability of symbols in the *User Timeline* visualization is only slightly less good. In both visual notations, enough visual variables are used to ensure that symbols that represent different constructs are easily distinguished. Discriminating between instances of a same construct that share many similar properties (e.g., two *feature-related events* that differ on the specific feature used, while happening at the same time in the same session) is significantly harder but is addressed by the tool-tip. Improving perceptual discriminability would require improving the visual expressiveness of the visualizations (see below).

Semiotic Transparency Both visualizations perform only mildly with regards to this principle. The use of colored lines and circles as the main shapes in the notation comes from the visualization template that we based our design on. While it provides a uniform look to all visualizations based on it, the template leaves very little space for implementing mechanisms that would improve the transparency of the notation. Furthermore, as noted by a respondent of the questionnaire, the choice of colors may give a (wrong) cue to the meaning of events (e.g., red or orange usually refer to warning or errors). This can be fixed by using more neutral colors for events. Note that the legend should prevent this misinterpretation. The use of icons to better represent construct types and event types could help improve the semiotic transparency of our visualizations, but would distance them from the original templates.

Complexity Management Our visualization can prove to be very complex in terms of *diagrammatic complexity* (i.e., the number of symbol instances displayed). Both visualizations are good at managing this complexity, firstly though the time selector. which can be used to limit the events shown within a time frame. This mechanism is included in the

default visualization template. The *Session Timeframe* visualization further manages this complexity by providing two filters, for both *event types* and *users*. The *User Timeline* notation uses instead hierarchy to divide the visualization into users and sessions. All those mechanisms are efficient and ensure that the notations are not overloaded with symbols.

Cognitive Integration Between the two visualizations, cognitive integration was not thought of at all. This is an obvious problem, as they are two different visualizations of the same data, and nothing is implemented to use them in conjunction. This originates from the technical framework that we are working with, which clearly never took this aspect into account. Within the *User Timeline* visualization - seen as a combination of session diagrams - little is done to help navigate between the sessions or the users. The sole *perceptual integration* mechanism is the user and session selection, which is done in arguably the most crude way. A more user-friendly diagram selection mechanism would improve this aspect. Furthermore, as it was noted by a respondent of the questionnaire, a "possibility to see a sample of all of the sessions, possibly filtered by the selected user" would be a great addition to the visualization. Effectively, this would provide a much needed *cognitive integration* mechanism.

Visual Expressiveness Both visualizations make a relatively good use of the graphic design space by using between five and six visual variables out of eight (shape, color, brightness, size, and horizontal and vertical position). The left-out variables are *texture* and *orientation*. The use of texture could significantly improve both visualizations if used in conjunction with colors as a form of redundant coding. This would fix the issue pointed out in the results, that color was used as the sole variable used to discriminate between event and construct types. Orientation would be hard to include in the visualizations as they are. Circles do not support it, and the lifespan lines would lose meaning. This highlights another problem which is that while shapes are used, the notations use only two of them: lines and circles. Increasing the range of shapes used would improve the notations and enable the use of orientation.

Dual Coding This principle is not used extensively by our visualizations. The main use of text to complement graphics is done through the tool-tip box. Which, as it was noted by a respondent of the questionnaire, "could be improved by formatting and removing unnecessary data". In practice, the tool-tip was thought of more as a debugging tool when designing the visualizations. We believe that improving it to become a real *annotation* would prove extremely useful. *Hybrid symbols* are used in few instances, but could also be added to help improve the perceptual discriminability of event instances.

Graphic Economy As both visualizations have a very limited visual vocabulary (coming from the low semantic complexity), their *graphic complexity* is already quite low. This is even improved by having symbol deficit and using combination of visual variables (good

visual expressiveness). We do not believe that it is necessary to look at improving the graphic economy of either visualizations.

Cognitive Fit The *representation medium* of our visualisations is fixed: a computer screen. This comes from the fact that a) the visualizations are generated by software, and b) are interactive. Hence adapting the visual notations to make them fit another medium is not an option. The real question from cognitive fit theory that applies here is the one of the *audience*. In our case, the target audience was always thought of as the developers of the software. This is inherent to what data we are trying to visualize, and has less to do with the visual syntax itself. We believe that the visual notation itself can be understood by both experts and novices and that in this case, having multiple visual dialects makes little sense.

6.1.3 Questionnaire

The questionnaire was answered by the two developers of Kactus2, which are the main target audience of our visualizations. The results show that the respondents found both visualizations *easy to understand*, and considered that they *could be used to assess the usage of Kactus2* and thus *improve the software* in some way. These responses are very encouraging, considering the limitations of the data itself (see below). Several ideas for improvements were proposed, suggesting that the developers indeed understood the potential applications of the visualizations in their work.

6.2 Research Questions

The interpretation of the results hereabove enables us to answer the original research questions.

Can the Unified Data Model for Software Engineering be used to efficiently model software usage data? The results of the ontological analysis have shown that our implementation of the UMSED, while incomplete with regards to the BWW-model, is able to represent most ontological concepts. More importantly, the semantic domain was found to be mostly clear, with the exception of the state as previously noted. The model is thus, according to our results, not perfect. But it does a satisfying job at representing the usage data collected from Kactus2. We argue that the presented model is a sufficiently sound basis for modeling software usage data, as it proved to be the case for our visualizations.

Can the framework presented be used to develop insightful visualizations of software usage data? The results of the syntactic analysis indicate that both visualizations are overall cognitively efficient. But also pointed out areas where major improvements

in the visual syntax could be made. Hence we argue that the technical framework and visualization templates are a good basis for developing good visual notations of software usage data. Furthermore, the results of the questionnaire strongly suggest that these visualizations may indeed provide useful insight into the usage of Kactus2. We argue that, given carefully collected usage data, the presented framework can help in developing acute visualizations of software usage data.

Ultimately and considering the limitations detailed below, we argue that the presented data model and visualization framework is indeed suitable for developing software usage visualizations that provide useful insight into the usage of Kactus2.

6.3 Limitations

Data source One major limitation comes from the data itself. Making a good visualization without good data is a complicated task. Our data source was far from being perfect. First of all we had no control over the data, as it had been collected back in 2017 (data collection itself was out of the scope of this thesis). The collection of this data was not motivated by either an explanatory nor a deployment analysis as it usually is the case. Rather, it was done with the main purpose of trying out a data collection technique. The data that would be collected was thus only secondary to the process. Secondly, the data was collected over a short period of time (1 month), with a limited set of participants (16 users) which were performing similar predefined tasks (laboratory exercises). We are thus far from real-world usage data of the software. Thirdly, the implementation of the data collection was not perfect either. For example, some events are sometimes logged twice, not all events are properly logged (like closing of a session), and most importantly the timestamps are only accurate to the second - resulting in many seemingly concurrent events. All these lead to an imperfect data source, that may not be an accurate representation of the reality that the visualizations pretend to display.

Semantic domain Both the ontological analysis and the first principle of the Physics of Notations (semiotic clarity) rely on a preexisting definition of the semantic domain of the visualization. We had no formal ground for defining such a domain, and it was constructed by trial and error until it seemed accurate enough. We believe that such a lack of a theoretical basis for defining the semantic constructs that form the domain is a potential source of bias in our subsequent analysis.

Questionnaire The questionnaire was answered by the two developers of Kactus2. This is obviously a very small sample of respondents to make generalizations out of it. Moreover, we have no expertise in designing academic questionnaires, and many biases may be present in the questions themselves, and no pretesting of the questionnaire was performed. Finally, knowing that the questionnaire would be used to assess the quality of

the work of this thesis, the respondents might have been kinder in their evaluations than otherwise. This suggests caution in the interpretation of the responses.

6.4 Future work

Future work should firstly focus on addressing the aforementioned limitations of this thesis. One direction would be to put the data model through further testing with large-scale, real-world usage data. Future research on usage data visualization should also aim to integrate proper user feedback through better means than a questionnaire, if possible. Regarding the visualization templates, the issues discussed could be resolved by future versions of the technical framework. Finally, usage data coming from different data sources could be combined using the UMSED, to try and provide even more comprehensive visualizations of the data.

7 CONCLUSION

This thesis aimed at investigating the adequacy of the UMSED (Unified Model for Software Engineering Data) and its technical framework for developing visualizations of software usage data. We started by taking logs from the software Kactus2, parsing and formatting them to fit the database used in the framework. The logs, containing timestamped recordings of user interactions with the software, had been previously collected in 2017. Once the data was completely integrated, we designed and implemented two visualizations by adding onto the original code base of the framework. The first visualization aimed at comparing the evolution of user-sessions in the software. The second focused on analysing usage of the software by each user.

On both visualizations, we performed a semantic (ontological) analysis and a syntactic analysis (using the Physics of Notations' principles). Furthermore, the visualizations were evaluated by the two developers of the software through a questionnaire.

We found that the presented data model and technical framework presented were adequate for visualizing the usage data from Kactus2. The visualizations performed well both in terms of semantic expressiveness and cognitive effectiveness. Moreover, they appeared to be insightful, providing important information regarding the use of the software, and easily understandable by the developers.

Further research should focus on generalizing those results by using multiple data sources and performing large-scale user evaluation of such type of visualizations. In general, the field of software usage data visualization is still very much unexplored. Our work thus contributes to expanding the amount of scientific research on the subject.

We hope that it will inspire researchers to further study and evaluate the usability of software usage data visualizations. Similarly, it should encourage companies to invest into the implementation of visual notations of usage data, in order to improve their software processes and products.

REFERENCES

- [1] R. Atterer, M. Wnuk, and A. Schmidt. Knowing the user's every move: user activity tracking for website usability evaluation and implicit interaction. In: *Proceedings of the 15th international conference on World Wide Web*. ACM. 2006, 203–212.
- [2] A. Banerjee, T. Bandyopadhyay, and P. Acharya. Data analytics: Hyped up aspirations or true potential? In: *Vikalpa* 38.4 (2013), 1–12.
- [3] A. Begel and T. Zimmermann. Analyze this! 145 questions for data scientists in software engineering. In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, 12–23.
- [4] J. Bertin, W. J. Berg, and H. Wainer. *Semiology of graphics: diagrams, networks, maps*. Vol. 1. 0. University of Wisconsin press Madison, 1983.
- [5] P. A. Brooks and A. M. Memon. Automated GUI testing guided by usage profiles. In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM. 2007, 333–342.
- [6] R. E. Bucklin and C. Sismeiro. A model of web site browsing behavior estimated on clickstream data. In: *Journal of marketing research* 40.3 (2003), 249–267.
- [7] M. Bunge. *Treatise on basic philosophy: Ontology I: the furniture of the world*. Vol. 3. Springer Science & Business Media, 1977.
- [8] M. Bunge. *Treatise on basic philosophy: Ontology II: A world of systems*. Vol. 4. D. Reidel, 1979.
- [9] R. P. Buse and T. Zimmermann. Analytics for software development. In: *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM. 2010, 77–80.
- [10] R. P. Buse and T. Zimmermann. Information needs for software development analytics. In: *Proceedings of the 34th international conference on software engineering*. IEEE Press. 2012, 987–996.
- [11] P. K. Chittimalli and V. Shah. GEMS: a generic model based source code instrumentation framework. In: *2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST 2012)*. IEEE. 2012, 909–914.
- [12] T. H. Davenport, J. G. Harris, and R. Morison. *Analytics at work: Smarter decisions, better results*. Harvard Business Press, 2010.
- [13] S. Diehl. *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.
- [14] S. Elbaum, S. Karre, and G. Rothermel. Improving web application testing with user session data. In: *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society. 2003, 49–59.

- [15] S. Elbaum, G. Rothermel, S. Karre, and M. Fisher II. Leveraging user-session data to support web application testing. In: *IEEE Transactions on Software Engineering* 31.3 (2005), 187–202.
- [16] A. Fabijan, H. H. Olsson, and J. Bosch. Time to Say'Good Bye': Feature Lifecycle. In: *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*. IEEE. 2016, 9–16.
- [17] A. Gehlert and W. Esswein. Toward a formal research framework for ontological analyses. In: *Advanced Engineering Informatics* 21.2 (2007), 119–131.
- [18] N. Genon, P. Heymans, and D. Amyot. Analysing the cognitive effectiveness of the BPMN 2.0 visual notation. In: *International Conference on Software Language Engineering*. Springer. 2010, 377–396.
- [19] H. H. Goldstine, J. Von Neumann, and J. Von Neumann. Planning and coding of problems for an electronic computing instrument. In: (1947).
- [20] N. Goodman and N. A. GOODMAN. *Languages of art: An approach to a theory of symbols*. Hackett publishing, 1968.
- [21] O. Hylli, A.-L. Mattila, and K. Systä. Collecting issue management data for analysis with a unified model and API descriptions. In: *SPLST*. 2015, 251–265.
- [22] A. Kamppi, L. Matilainen, J.-M. Maatta, E. Salminen, T. D. Hamalainen, and M. Hannikainen. Kactus2: Environment for embedded product development using ip-xact and mcapi. In: *Digital System Design (DSD), 2011 14th Euromicro Conference on*. IEEE. 2011, 262–265.
- [23] A. Kamppi, E. Pekkarinen, J. Virtanen, J.-M. Määttä, J. Järvinen, L. Matilainen, M. Teuho, and T. D. Hämäläinen. Kactus2: A graphical EDA tool built on the IP-XACT standard. In: *The Journal of Open Source Software* 2 (2017).
- [24] A. Kaushik. *Web analytics 2.0: The art of online accountability and science of customer centrality*. John Wiley & Sons, 2009.
- [25] J. Kim, J. Hahn, and H. Hahn. How do we understand a system with (so) many diagrams? Cognitive integration processes in diagrammatic reasoning. In: *Information Systems Research* 11.3 (2000), 284–303.
- [26] J. H. Larkin and H. A. Simon. Why a diagram is (sometimes) worth ten thousand words. In: *Cognitive science* 11.1 (1987), 65–100.
- [27] F. Lautenschlager and M. Ciolkowski. Making Runtime Data Useful for Incident Diagnosis: An Experience Report. In: *International Conference on Product-Focused Software Process Improvement*. Springer. 2018, 422–430.
- [28] J. Lee, M. Podlaseck, E. Schonberg, and R. Hoch. Visualization and analysis of clickstream data of online stores for understanding web merchandising. In: *Data Mining and Knowledge Discovery* 5.1-2 (2001), 59–84.
- [29] S. Marciuska and P. Abrahamsson. Feature Usage Explorer: Usage Monitoring and Visualization Tool in HTML5 Based Applications. In: *Journal of Open Research Software* 1.1 (2013).

- [30] S. Marciuska, C. Gencel, and P. Abrahamsson. Exploring how feature usage relates to customer perceived value: A case study in a startup company. In: *International Conference of Software Business*. Springer. 2013, 166–177.
- [31] S. Marciuska, C. Gencel, and P. Abrahamsson. Automated feature identification in web applications. In: *International Conference on Software Quality*. Springer. 2014, 100–114.
- [32] S. Marciuska, C. Gencel, X. Wang, and P. Abrahamsson. Feature usage diagram for feature reduction. In: *International Conference on Agile Software Development*. Springer. 2013, 223–237.
- [33] J. Matejka, T. Grossman, and G. Fitzmaurice. Patina: Dynamic heatmaps for visualizing application usage. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2013, 3227–3236.
- [34] A.-L. Mattila, P. Ihanola, T. Kilamo, A. Luoto, M. Nurminen, and H. Väätäjä. Software visualization today: Systematic literature review. In: *Proceedings of the 20th International Academic Mindtrek Conference*. ACM. 2016, 262–271.
- [35] A.-L. Mattila, T. Lehtonen, H. Terho, T. Mikkonen, and K. Systä. Mashing up software issue management, development, and usage data. In: *Proceedings of the Second International Workshop on Rapid Continuous Software Engineering*. IEEE Press. 2015, 26–29.
- [36] A.-L. Mattila, A. Luoto, H. Terho, O. Hylli, O. Sievi-Korte, and K. Systsä. Unified model for software engineering data. In: *2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. IEEE. 2015, 150–154.
- [37] A.-L. Mattila, K. Systä, O. Sievi-Korte, M. Leppänen, and T. Mikkonen. Discovering Software Process Deviations Using Visualizations. In: *International Conference on Agile Software Development*. Springer. 2017, 259–266.
- [38] R. E. Mayer and R. Moreno. Nine ways to reduce cognitive load in multimedia learning. In: *Educational psychologist* 38.1 (2003), 43–52.
- [39] T. Menzies and T. Zimmermann. Software analytics: so what? In: *IEEE Software* 4 (2013), 31–37.
- [40] G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. In: *Psychological review* 63.2 (1956), 81.
- [41] D. Moody. The “physics” of notations: toward a scientific basis for constructing visual notations in software engineering. In: *IEEE Transactions on Software Engineering* 35.6 (2009), 756–779.
- [42] A. Newell, H. A. Simon, et al. *Human problem solving*. Vol. 104. 9. Prentice-Hall Englewood Cliffs, NJ, 1972.
- [43] H. H. Olsson and J. Bosch. Post-deployment data collection in software-intensive embedded products. In: *Continuous software engineering*. Springer, 2014, 143–154.
- [44] A. L. Opdahl and B. Henderson-Sellers. Ontological evaluation of the UML using the Bunge–Wand–Weber model. In: *Software and systems modeling* 1.1 (2002), 43–67.

- [45] A. Paivio. *Mental representations: A dual coding approach*. Vol. 9. Oxford University Press, 1990.
- [46] A. Patrushev. Analysis of software engineering data with self-organizing maps. In: (2016).
- [47] S. Sampath, V. Mihaylov, A. Souter, and L. Pollock. A scalable approach to user-session based testing of web applications through concept analysis. In: *Proceedings of the 19th IEEE international conference on Automated software engineering*. IEEE Computer Society. 2004, 132–141.
- [48] R. Schuette and T. Rotthowe. The guidelines of modeling—an approach to enhance the quality in information models. In: *International Conference on Conceptual Modeling*. Springer. 1998, 240–254.
- [49] H. van der Schuur, S. Jansen, and S. Brinkkemper. A reference framework for utilization of software operation knowledge. In: *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on*. IEEE. 2010, 245–254.
- [50] T. M. Shaft and I. Vessey. The role of cognitive fit in the relationship between software comprehension and modification. In: *Mis Quarterly* (2006), 29–55.
- [51] G. Shanks, E. Tansley, and R. Weber. Using ontology to validate conceptual models. In: *Communications of the ACM* 46.10 (2003), 85–89.
- [52] K. Siau. Informational and computational equivalence in comparing information modeling methods. In: *Journal of Database Management (JDM)* 15.1 (2004), 73–86.
- [53] S. Sprenkle, S. Sampath, E. Gibson, L. Pollock, and A. Souter. An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. In: *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE. 2005, 587–596.
- [54] S. Suonsyrjä. Eeny, Meeny, Miny, Mo... In: *International Conference on Agile Software Development*. Springer. 2017, 52–67.
- [55] S. Suonsyrjä, L. Hokkanen, H. Terho, K. Systä, and T. Mikkonen. Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development? In: *Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA), 2016 Joint Conference of the International Workshop on*. IEEE. 2016, 139–147.
- [56] S. Suonsyrjä, K. Systä, T. Mikkonen, and H. Terho. Collecting Usage Data for Software Development: Selection Framework for Technological Approaches. In: *SEKE*. 2016, 114–119.
- [57] P. Tyrväinen, M. Saarikallio, T. Aho, T. Lehtonen, and R. Paukeri. Metrics framework for cycle-time reduction in software value creation. In: *ICSEA 2015: The Tenth International Conference on Software Engineering Advances, ISBN 978-1-61208-438-1*. IARIA. 2015.
- [58] D. Van Der Linden and I. Hadar. A Systematic Literature Review of Applications of the Physics of Notation. In: *IEEE Transactions on Software Engineering* (2018).

- [59] I. Vessey. Cognitive fit: A theory-based analysis of the graphs versus tables literature. In: *Decision Sciences* 22.2 (1991), 219–240.
- [60] I. Vessey and D. Galletta. Cognitive fit: An empirical study of information acquisition. In: *Information systems research* 2.1 (1991), 63–84.
- [61] Y. Wand and R. Weber. An ontological model of an information system. In: *IEEE transactions on software engineering* 16.11 (1990), 1282–1292.
- [62] Y. Wand and R. Weber. On the ontological expressiveness of information systems analysis and design grammars. In: *Information Systems Journal* 3.4 (1993), 217–237.
- [63] W. Winn. Encoding and retrieval of information in maps and diagrams. In: *IEEE Transactions on Professional Communication* 33.3 (1990), 103–107.

A ACTIONS RECORDED IN THE LOGS

The log¹ of Kactus2 is comprised of 4627 entries shared between 64 sessions created by 16 users. The average number of entries per session is 72,296875. The different types of entries recorded are listed here: *Clicked on...*

- About
- Configure Library
- Default
- Exit
- Help
- Interconnection Tool
- Locked / Unlocked
- Makefile Generator
- MCAPI Code Generator
- New
- Refresh
- Refresh Library
- Save / Save As / Save All
- Select Tool
- Settings
- Undo
- View Library Integrity Report
- Visible Windows

And the following:

- Session started
- Document X opened / closed / locked / unlocked
- Help page Y opened

¹Complete logs can be found at <https://github.com/coin-quin/vis-a-vis/blob/master/datasources/log/log.txt>

B BWW ONTOLOGICAL CONCEPTS

The following tables (B.1 to B.9) present the basic concepts of the BWW-model as described in [44]. Note: BWW refers to Bunge-Wand-Weber [61, 62], to distinguish the ontological concepts from the semantic constructs of the semantic domain (see appendix C).

Ontological Construct	Description
BWW-thing	The elementary unit in our ontological model. The real world is made up of things.
BWW-property [of a thing], BWW-property of a particular	Things possess properties, we know about things in the world via their properties.
BWW-property function [of a thing]	A property is modeled via a function that maps the thing into some value. A BWW-property function represents how a property changes over time.
BWW-codomain [of a property function]	The set of values into which the function that stands for the property of a thing maps the thing.
BWW-intrinsic property [of a thing]	A property that is inherently a property of an individual thing.
BWW-mutual property [of two or more things]	A property that is meaningful only in the context of two or more things. A property is either intrinsic or mutual, exclusively.
BWW-complex property [of a thing]	A complex BWW-property comprises other properties, which may themselves be complex.

Table B.1. Basic concepts in the BWW-model: Things and properties.

Ontological Construct	Description
BWW-law property [of a thing]	Properties can be restricted by laws relating to one or several properties.
BWW-natural law property	Natural laws are established by nature.
BWW-human law property	Some laws are human-made artifacts. 1) Events and processes may sometimes violate human laws, but not natural ones. 2) A law is either natural or human, exclusively.

Table B.2. Basic concepts in the BWW-model: Natural and human laws.

Ontological Construct	Description
BWW-class [of things]	A set of things that can be defined by their possessing a particular set of properties. All groups of BWW-properties that are possessed by at least one BWW-thing define a BWW-class.
BWW-natural kind [of things]	A natural kind is defined by a set of properties and the laws connecting them.
BWW-characteristic property [of a class or natural kind], BWW-property in general	A property (in general) that defines a class or natural kind. If the property is a law, it defines a natural kind, not a class.
BWW-subclass [of things]	A set of things that can be defined via their possessing the set of properties in a class plus an additional set of properties.
Subkind (sub-natural kind) [of things]	A set of things that can be defined via their possessing the set of properties and laws in a natural kind plus an additional set of properties and the laws connecting them.
Natural kind/sub-kind relationship	The relationship between the kind and subkind in the above definition.

Table B.3. Basic concepts in the BWW-model: Classes and natural kinds.

Ontological Construct	Description
BWW-state [of a thing]	The vector of values for all property functions of a thing.
BWW-history [of a thing]	The chronologically ordered states that a thing traverses in time.
BWW-event [in a thing]	A change of state of a thing. It is effected via a transformation (see below).
BWW-process [in a thing or system thing]	An intrinsically ordered sequence of events on, or states of, a thing. Processes are either chains or trees of events.
BWW-transformation [of a thing]	A mapping from a domain comprising states to a codomain comprising states.

Table B.4. Basic concepts in the BWW-model: States, events and transformations.

Ontological Construct	Description
BWW-state law [of a thing]	A property that restricts the values of the property functions of a thing to a subset that is deemed lawful because of natural laws or human laws.
BWW-transformation law [of a thing]	Events are governed by transformation laws that define the allowed changes of state.

Table B.5. Basic concepts in the BWW-model: State and transformation laws.

Ontological Construct	Description
BWW-external event [in a thing, subsystem or system]	An event that arises in a thing, subsystem or system by virtue of the action of some thing in the environment of the thing, subsystem or system. The before-state of an external event is always stable. The after-state may be stable or unstable.
BWW-internal event [in a thing, subsystem or system]	An event that arises in a thing, subsystem or system by virtue of lawful transformations in the thing, subsystem or system. The before-state of an internal event is always unstable. The after-state may be stable or unstable.
BWW-stable state [of a thing]	A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event).
BWW-unstable state [of a thing]	A state that will be changed into another state by virtue of the action of transformation in the system.

Table B.6. Basic concepts in the BWW-model: Internal/external events and stable/unstable states.

Ontological Construct	Description
BWW-conceivable state space [of a thing]	The set of all states that the thing may ever assume.
BWW-possible state space [of a thing]	The space of states that are possible given our understanding of the laws of nature.
BWW-lawful state space [of a thing]	The set of states of a thing that comply with the state laws of the thing.
BWW-conceivable event space [of a thing]	The set of all possible events that can occur in the thing.
BWW-lawful event space [of a thing]	The set of all events in a thing that are lawful, because (a) nature permits them to occur, and (b) there are no human laws that denote them as unlawful.

Table B.7. Basic concepts in the BWW-model: State and event spaces.

Ontological Construct	Description
BWW-coupling [of things], BWW-acting on [another thing]	A thing acts on another thing if its existence affects the history of the other thing. The two things are said to be coupled.
BWW-binding mutual property, BWW-direct acting on	A thing acts directly on one or more other things when the former thing changes a BWW-binding mutual property they all possess.
BWW-coupled event	When an event in one thing changes a BWW-binding mutual property and thereby causes an external event in another thing.

Table B.8. Basic concepts in the BWW-model: Coupling.

Ontological Construct	Description
BWW-composite thing	A composite thing may be made up of other things (composite or primitive). Things can be combined to form a composite thing.
BWW-component thing	A BWW-thing in the composition of a composite thing.
BWW-whole-part relation [between things]	The property of being in the composition of another thing or, complementary, of having another thing as a component.
BWW-resultant property [of a composite thing]	A property of a composite thing that belongs to a component thing.
BWW-emergent property [of a composite thing]	A property of a composite thing that does not belong to a component thing.
BWW-system [of things]	A set of things is a system if, for any bi-partitioning of the set, couplings exist among things in the two subsets.
BWW-system composition	The things in the system, i.e., its component things.
BWW-system environment	Things that are not in the system but interact with things in the system.
BWW-system structure	The set of couplings that exist among things in the system and things in the environment of the system.
BWW-subsystem	A system whose composition and structure are subsets of the composition and structure of another system.
BWW-system decomposition	A set of subsystems such that every component in the system is either one of the subsystems in the decomposition or is included in the composition of one of the subsystems in the decomposition.
BWW-level structure	Defines a partial order over the subsystems in a decomposition to show which subsystems are components of other subsystems or the system itself.

Table B.9. Basic concepts in the BWW-model: Composites and systems.

C VISUALIZATION SEMANTIC CONSTRUCTS

The following tables (C.1 to C.4) describe the semantic domain of our visualizations. That is, the set of semantic constructs that are described by or can be inferred from the data model. Note: SEM refers in this context to *semantic*, to distinguish the semantic constructs from the ontological concepts of the BWW-model (see appendix B).

Semantic Construct	Description
SEM-entity	Entities are the semantic building blocks. That includes SEM-objects and SEM-actions.
SEM-entity property	Entities have properties, such as a <i>unique identifier</i> .

Table C.1. Basic concepts in the semantic-model: Entities.

Semantic Construct	Description
SEM-action	Actions on the SEM-objects. That includes SEM-events and SEM-state changes.
SEM-event	Events are actions performed by a SEM-user.
SEM-state change	A state change in a SEM-object is triggered by a SEM-event performed on that object.
SEM-action property	Actions possess properties, such as <i>time</i> and <i>date</i> .
SEM-event property	Events possess properties (such as <i>type</i> for example).
SEM-state change property	State changes possess properties, such as <i>from</i> , <i>to</i> .

Table C.2. Basic concepts in the semantic-model: Actions.

Semantic Construct	Description
SEM-object	Objects are what the data model used calls as <i>constructs</i> . That includes SEM-users, SEM-sessions, SEM-documents, and SEM-help pages.
SEM-user	Users are the humans that interact with the software. They perform actions on other objects. A user has a <i>name</i> (unique identifier).
SEM-session	Sessions are the underlying structure of the user behavior. They are created and terminated by the SEM-user. It is within a session that other objects (SEM-documents, SEM-help pages) exist, and that actions occur. A session has a <i>name</i> (unique identifier).
SEM-document	Documents are files opened, closed, owned and in general used by a SEM-user within a SEM-session. A document has a <i>file name</i> and a <i>hash</i> (unique identifier).
SEM-help page	Help pages are html files opened by a SEM-user within a SEM-session. A help page has a <i>file name</i> and a <i>hash</i> (unique identifier).
SEM-object property	Objects possess properties.
SEM-user property	Users possess properties, such as <i>name</i> .
SEM-session property	Sessions possess properties, such as <i>name</i> or <i>timeframe</i> .
SEM-document property	Documents possess properties, such as <i>name</i> , <i>hash</i> or <i>timeframe</i> .
SEM-help page property	Help pages possess properties, such as <i>name</i> , <i>hash</i> or <i>timeframe</i> .

Table C.3. Basic concepts in the semantic-model: Objects.

Semantic Construct	Description
SEM-relation between SEM-entities	Two or more entities can be linked by a relation.
SEM-relation between SEM-objects	Two or more objects can be linked by a relation.
SEM-relation between SEM-object and SEM-action	An object can be linked to an event or a state change by a relation. An event or state change is always linked to an object by a relation.
SEM-relation between SEM-actions	Two events can be linked by a relation. An event can be lined to a single state change by a relation. A state change is always lined to an event by a relation.

Table C.4. Basic concepts in the semantic-model: Relations.

D REPRESENTATION MAPPING

The following tables (D.1 to D.9) present the representation mapping between the ontological concepts and the semantic constructs. Note: BWW refers to Bunge-Wand-Weber [61, 62], to distinguish the ontological concepts from the semantic constructs of the semantic domain, referred to as SEM.

Ontological Concept	Semantic Construct
BWW-thing	Instance of a SEM-entity.
BWW-property [of a thing], BWW-property of a particular	SEM-entity property.
BWW-property function [of a thing]	The state of SEM-objects may change over time. This is represented by their <i>lifespan</i> .
BWW-codomain [of a property function]	List of states of a SEM-object.
BWW-intrinsic property [of a thing]	SEM-event property. Some SEM-object property (such as state).
BWW-mutual property [of two or more things]	SEM-state change property. Some SEM-object property.
BWW-complex property [of a thing]	Some SEM-object properties are complex properties.

Table D.1. Representation mapping of the BWW-model: Things and properties.

Ontological Concept	Semantic Construct
BWW-law property [of a thing]	Rules imposed on SEM-entities and SEM-relations.
BWW-natural law property	Rules given by the data model.
BWW-human law property	Rules given by the data collection.

Table D.2. Representation mapping of the BWW-model: Natural and human laws.

Ontological Concept	Semantic Construct
BWW-class [of things]	SEM-objects, SEM-actions.
BWW-natural kind [of things]	-
BWW-characteristic property [of a class or natural kind], BWW-property in general	SEM-object type: user, session, document, help page. SEM-action type: event, state change.
BWW-subclass [of things]	SEM-user, -session, -document, -help page are subclasses of SEM-object. SEM-event, -state change are subclasses of SEM-action.
Subkind (sub-natural kind) [of things]	-
Natural kind/sub-kind relationship	-

Table D.3. Representation mapping of the BWW-model: Classes and natural kinds.

Ontological Concept	Semantic Construct
BWW-state [of a thing]	State of SEM-objects (it is a SEM-object property).
BWW-history [of a thing]	List of SEM-state changes linked to a SEM-object.
BWW-event [in a thing]	SEM-state change.
BWW-process [in a thing or system thing]	Lifespan of a SEM-object.
BWW-transformation [of a thing]	Mapping of SEM-state changes' property (from, to).

Table D.4. Representation mapping of the BWW-model: States, events and transformations.

Ontological Concept	Semantic Construct
BWW-state law [of a thing]	Collection of possible states of SEM-objects.
BWW-transformation law [of a thing]	Collection of possible SEM-state changes of SEM-objects.

Table D.5. Representation mapping of the BWW-model: State and transformation laws.

Ontological Concept	Semantic Construct
BWW-external event [in a thing, sub-system or system]	Some SEM-events: <i>session start, document open, help page open</i> .
BWW-internal event [in a thing, sub-system or system]	The other SEM-events.
BWW-stable state [of a thing]	<i>Closed</i> states of SEM-objects.
BWW-unstable state [of a thing]	Other states of SEM-objects.

Table D.6. Representation mapping of the BWW-model: Internal/external events and stable/unstable states.

Ontological Concept	Semantic Construct
BWW-conceivable state space [of a thing]	All states of SEM-objects.
BWW-possible state space [of a thing]	States of SEM-objects allowed by the model.
BWW-lawful state space [of a thing]	States of SEM-objects that are coherent with logic.
BWW-conceivable event space [of a thing]	SEM-actions.
BWW-lawful event space [of a thing]	SEM-actions that are coherent with logic.

Table D.7. Representation mapping of the BWW-model: State and event spaces.

Ontological Concept	Semantic Construct
BWW-coupling [of things], BWW-acting on [another thing]	SEM-relations between entities.
BWW-binding mutual property, BWW-direct acting on	SEM-relation between SEM-event and SEM-state change.
BWW-coupled event	SEM-relation between SEM-action and SEM-object.

Table D.8. Representation mapping of the BWW-model: Coupling.

Ontological Concept	Semantic Construct
BWW-composite thing	-
BWW-component thing	-
BWW-whole-part relation [between things]	-
BWW-resultant property [of a composite thing]	-
BWW-emergent property [of a composite thing]	-
BWW-system [of things]	The whole data structure of SEM-entities and SEM-relations.
BWW-system composition	Combination of SEM-entities and SEM-relations.
BWW-system environment	-
BWW-system structure	Structure of the data model.
BWW-subsystem	A SEM-session and the related SEM-objects and SEM-actions.
BWW-system decomposition	-
BWW-level structure	Rules that dictate the subsystem.

Table D.9. Representation mapping of the BWW-model: Composites and systems.

E INTERPRETATION MAPPING

The following tables (E.1 to E.4) present the interpretation mapping between the semantic constructs and the ontological concepts. Note: SEM refers in this context to *semantic*, to distinguish the semantic constructs from the ontological concepts, referred to as BWW.

Semantic Construct	Ontological concept
SEM-entity	BWW-thing.
SEM-entity property	BWW-property of a thing.

Table E.1. Interpretation mapping of the semantic-model: Entities.

Semantic Construct	Ontological concept
SEM-action	BWW-class of things.
SEM-event	BWW-subclass of SEM-action.
SEM-state change	BWW-subclass of SEM-action.
SEM-action property	BWW-property of SEM-action.
SEM-event property	BWW-intrinsic property of SEM-event.
SEM-state change property	BWW-mutual property of SEM-state change and the SEM-object whose state changes.

Table E.2. Interpretation mapping of the semantic-model: Actions.

Semantic Construct	Ontological concept
SEM-object	BWW-class of things.
SEM-user	BWW-subclass of SEM-object.
SEM-session	BWW-subclass of SEM-object.
SEM-document	BWW-subclass of SEM-object.
SEM-help page	BWW-subclass of SEM-object.
SEM-object property	BWW-property of SEM-object.
SEM-user property	BWW-property of SEM-user. The user <i>name</i> is a BWW-intrinsic property.
SEM-session property	BWW-property of SEM-session. The session <i>name</i> is a BWW-intrinsic property.
SEM-document property	BWW-property of SEM-document. The document <i>name</i> is a BWW-intrinsic property. The document <i>hash</i> is a BWW-mutual property of both the document and the session.
SEM-help page property	BWW-property of SEM-help page. The help page <i>name</i> is a BWW-intrinsic property. The help page <i>hash</i> is a BWW-mutual property of both the help page and the session.

Table E.3. Interpretation mapping of the semantic-model: Objects.

Semantic Construct	Ontological concept
SEM-relation between SEM-entities	BWW-coupling of things, BWW-acting on another thing.
SEM-relation between SEM-objects	BWW-coupling of things.
SEM-relation between SEM-object and SEM-action	BWW-coupling of SEM-actions, BWW-acting on SEM-objects.
SEM-relation between SEM-actions	BWW-coupling of SEM-events, BWW-acting on SEM-state changes.

Table E.4. Interpretation mapping of the semantic-model: Relations.