Analyzing web applications by visualizing their state-flow graphs

Master’s Thesis, October 18, 2013

Jelle Fresen
Analyzing web applications by visualizing their state-flow graphs

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Jelle Fresen
born in Rotterdam, The Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl
To my father
Analyzing web applications by visualizing their state-flow graphs

Author: Jelle Fresen
Student id: 1174479
Email: j.fresen@student.tudelft.nl

Abstract

In today's society, it is impossible to imagine life without web applications such as our webmail client or online word processors. These applications are becoming increasingly complex, which makes it hard to maintain a thorough understanding of the application by their developers. Therefore, a need has arisen for tools that support the comprehension of web applications.

This thesis builds upon previous work in which a crawler has been created that crawls a web application and generates its state-flow graph. We have proposed several concepts for the visualization of this state-flow graph to make it more comprehensible, such as choosing a dynamic graph drawing algorithm and applying clustering. We also integrated a visualization of test results into the state-flow graph to facilitate the locating of errors.

These proposals have been implemented in a proof of concept tool called Graphicrawl. Using this tool as context in a contextual interview, we then performed an evaluation study to verify the usefulness of the proposed concepts and gathered feedback to improve their effectiveness. The results indicate that Graphicrawl has the potential to support comprehension and several pointers for improvement have been identified.

Thesis Committee:
Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: Dr. A.E. Zaidman, Faculty EEMCS, TU Delft
Committee Member: Drs. P.R. van Nieuwenhuizen, Faculty EEMCS, TU Delft
Preface

Before you lies the apotheosis of my graduation project carried out to finally reach my Master’s degree at the Delft University of Technology. It has been a tumultuous time for me and during that time, the list of people I want to thank for helping me has grown too big to list here. However, to some of them I owe so much, that I’d like to express my gratitude in here.

Thank you Martin Pinzger, for taking me under your supervision during an all time low in my academic time and reaching out for me. Thank you especially, Andy Zaidman, for your never ending support and believe in me, but most of all for your patience with me and for getting me back on track time after time again. Thank you Nienke, for keeping me in line, supporting me in every way you could and always believing in me. Thank you Herman and Ingmar, for always being there for me. And finally, thanks to my lab mates with whom I could discuss my problems, have a good laugh and helped me get organized.

Jelle Fresen
Delft, the Netherlands
October 18, 2013
# Contents

Preface iii  

Contents v  

List of Figures vii  

1 Introduction 1  
1.1 Research Motivation ........................................ 1  
1.2 Research Questions and Approach .......................... 2  
1.3 Outline ................................................................... 3  

2 Background and Related Work 5  
2.1 Program Comprehension ........................................ 6  
2.2 Web Applications ................................................ 9  
2.3 Solution Methods ................................................ 10  
2.4 Available Tools .................................................. 18  

3 Approach 23  
3.1 Proposed concepts .............................................. 23  
3.2 Graphicrawl tool .................................................. 27  
3.3 Summary ............................................................ 32  

4 Implementation 33  
4.1 Graph architecture ............................................... 33  
4.2 Clustering algorithm .............................................. 35  
4.3 Test results ........................................................... 38  
4.4 Custom components .............................................. 40  

5 Evaluation Design 43  
5.1 Goal ................................................................. 43  
5.2 Evaluation method ............................................... 43  
5.3 Test setup ............................................................ 44  
5.4 Pilot study ........................................................... 48  
5.5 Participant selection .............................................. 48
CONTENTS

5.6 Evaluation study .............................................. 49

6 Evaluation Results ............................................. 51
  6.1 Participant profile .......................................... 51
  6.2 Result analysis .............................................. 52
  6.3 Threats to validity .......................................... 62

7 Conclusions and Future Work ................................. 65
  7.1 Conclusions .................................................. 65
  7.2 Contributions ............................................... 67
  7.3 Future work .................................................. 68

Bibliography ...................................................... 71

A Evaluation Documents ......................................... 77
  A.1 Knowledge Level ............................................ 78
  A.2 Graphicrawl Tutorial ....................................... 79
  A.3 Questionnaire ............................................... 83
## List of Figures

2.1 Model of the client side used by RE-RIA .............................................. 16
3.1 A hierarchical layout with many cross-links ................................. 25
3.2 The architecture of Graphicrawl .................................................. 28
3.3 The user interface of Graphicrawl (Graph View & Details View) ....... 29
3.4 The user interface of Graphicrawl (Test Result View) ...................... 31
3.5 The user interface of Graphicrawl (Test Result View) ...................... 31
4.1 Graphicrawl’s graph architecture ................................................. 34
4.2 An example of a clustering .......................................................... 36
4.3 The model for recording invariant results ...................................... 39
4.4 Graphicrawl’s clustering architecture .......................................... 41
6.1 Participants’ knowledge levels ..................................................... 52
6.2 Questionnaire results on interactive navigation ............................ 53
6.3 Questionnaire results on graph visualization ................................ 54
6.4 Questionnaire results on clustering ............................................. 54
6.5 Questionnaire results on usability ................................................. 56
6.6 Applicability of Graphicrawl to Sillito’s set of questions .............. 57
6.7 Questionnaire results on invariant results .................................... 58
6.8 Helpfulness of Graphicrawl with automated testing ...................... 59
6.9 Top 3 features of Graphicrawl .................................................... 60
6.10 Using Graphicrawl ................................................................. 61
Chapter 1

Introduction

In today's society, it is impossible to imagine life without web applications. We send mail through our webmail clients, write documents in online word processors, order products in webshops, draw diagrams in online editors, and so on. Even this very thesis was written with a web application\(^1\). The core technology on which these applications rely is AJAX, enabling a dynamic look and feel of an application within a web site.

All these marvels of technology have been written by teams of experienced web developers, who are never quite finished with their jobs: in most major web applications, new features are constantly being added and existing ones are updated. With this dynamic nature it is hard for web developers to maintain a thorough understanding of the application, even for the experienced ones [40]. On top of that, an ever changing system is prone to errors, demanding the repeated execution of test suites and analyzing their results. Test suites were often created manually, but research to their automatic generation has advanced greatly and tools can now be used effectively to generate test suites much larger than manually crafted ones [21].

Every now and then a team of developers is enriched with a fresh new addition. In some cases, this developer is already quite acquainted with the web application he will be working on, but in other cases he will have only a limited amount of knowledge about it. In those cases, much effort is required to bridge this gap of understanding, and tools that help him in this process can be a valuable addition.

In this thesis we will explore two possibilities of improvement of such tools as described above: one for a better comprehension of the interaction between different parts of the user interface, and one for an easier way to interpret the results of automatically generated user interface test suites.

1.1 Research Motivation

To accommodate developers with comprehending programs, much research has been done on program comprehension, as shown by the 20 year long history of the International Conference on Program Comprehension. However, within the research branch targeted at web applications, there has been done relatively few research [12], suggesting ample room for improvement.

\(^1\)www.sharelatex.com
One of the topics within the comprehension of web application covers the analysis of the navigation structure of web applications. This has been done by several authors (see Chapter 2.3), but so far only one of them succeeded to do this for AJAX enabled web applications: Mesbah et al. [36] created a crawler that automatically includes every part of the web application by identifying all clickable elements, Crawljax. This analysis yields a state-flow graph which can help web developers with comprehending the targeted application. However, the authors also pointed out that the concept of the state-flow graph suffers from the state explosion problem, quickly leading to a large and uncomprehensible graph, even for modestly sized applications. We believe that creating an interactive graph visualization and applying a graph clustering technique can restore the understandability of the graph when the state-flow graph in its entirety is too difficult to fathom.

Secondly, Crawljax has also been used to automatically generate regression test suites [45]. However, Robinson et al. noted that generated test suites are often difficult to understand [43], motivating the research towards increasing the comprehension of the test suites and their results. In 2003, Lanza described the successful implementation of polymetric views in their reverse engineering tool CodeCrawler [29], based on which we believe that the generated test suites from Roest et al. can become more understandable when they are integrated in the state-flow graph as a polymetric view, making it easier to locate the errors that were found with the test suites.

1.2 Research Questions and Approach

The above given research motivation has led us to put the following two research questions central in this thesis:

**RQ1** How can state-flow graphs of a web application be visualized to improve comprehension of the web application?

**RQ2** How can comprehension tools of web applications be enriched to support the locating of errors?

Since there can be no single answer to these questions, our goal will not be to give a definitive answer, but rather to collect knowledge on the dos and don’ts of different solutions to these questions. To achieve this goal, we will first propose several example solutions and implement them in a demo tool called Graphicrawl. We will then conduct an evaluation study with this tool to assess the effectiveness of the implementations, by trying to answer the following questions:

**RQ3** Does the state-flow graph visualization of Graphicrawl support developers in understanding web applications?

**RQ4** Does the visualization of DOM invariant results of Graphicrawl support developers with locating errors?

During the evaluation study, the focus shall also be laid on the more generic research questions RQ1 and RQ2 in order to identify other solutions then those that were implemented into Graphicrawl.
1.3 Outline

In the rest of this thesis you can read how the research was conducted that finally yields the answers to RQ1 till RQ4. For quick reference, below follows a quick description of each chapter.

In Chapter 2, the basic characteristics of web applications are explained and an overview of the related work on the comprehension of web applications is given, making a reasonable case that this thesis can contribute to this field of research.

Next, in Chapter 3, several concepts will be proposed that can improve the adequacy of the visualization of state-flow graphs, followed by a description of the tool and its features that we will implement, its architecture and an overview of the user interface of the tool.

Then, in Chapter 4, the implementation details and design choices made during the instantiation of the concepts can be found. This encompasses the more technical part of this thesis and is mostly interesting for developers wishing to extend or reuse the tool, covering the graph visualization, clustering technique, integration of the generated test suites and the extension points in the tool.

Chapter 5 describes the setup of the evaluation study. It includes the goal we had in mind, our considerations for different methods, the actual setup, the results of the pilot study, the way in which we have selected the participants and the circumstances in which the actual study was performed.

The results of the evaluation are written down in Chapter 6. They have been divided into two major parts, focussing on the two main questions of this thesis: how the users perceived the ability to support the comprehension of web applications, and how the users perceived the ability to support locating errors with the results of the generated test suites.

Finally, in Chapter 7 we draw our conclusions by answering the research questions and summarize the suggestions for future work that follows from the evaluation study.
Chapter 2

Background and Related Work

With the advent of Asynchronous JavaScript And XML (AJAX) around 2005 [24], a new trend was set for creating user friendly applications accessible through a browser. For example, rich email clients were developed such as GMail and Windows Live Hotmail, which were launched in 2004 and 2007 respectively. Many other such web applications (WAs) have been created since and some businesses have started to become dependent on them, demanding a continuous improvement in both features and stability. As a natural result, this led to the fact that WAs began to evolve and started suffering from the same difficulties as the maintenance of traditional desktop software. Amongst others, this meant that, for programmers, WAs became harder to understand, and thus harder to maintain and improve. This was already foreseen by Hassan and Holt even before the rise of AJAX, stating that “web applications are the legacy software of the future” [25]. Therefore, the need for program comprehension targeted on web applications grew swiftly, and with it the associated field of research: comprehension of web applications.

Many techniques from traditional program comprehension can be applied to the comprehension of web applications, yet WAs differ from desktop applications in some key aspects. First of all, web applications are usually heterogeneous in nature, i.e., they are composed of code in several programming languages, making its analysis inherently more difficult. Second, they can be dynamically configured, meaning that the code that is executed is often generated on the fly. Both these aspects make it imperative to use dynamic analysis, which, as will be seen, is a frequently used technique in the solutions presented in Section 2.3.

This chapter is structured as follows. Since the comprehension of web applications has its roots in program comprehension, Section 2.1 will look at several basic understandings from that field. Then in Section 2.2, a brief history of web applications will be given, in order to get a better understanding of the course and focus of research in the past. In Section 2.3, a number of results from literature on the comprehension of web applications will be listed. Finally, Section 2.4 enumerates and discusses several existing tools for the analysis of WAs.
2. BACKGROUND AND RELATED WORK

2.1 Program Comprehension

As software matures it is an almost universal fact that it steadily gains in complexity and incomprehensibility. This is for example due to the addition of new features, revisions from time to time to the structure of the code, bug fixes that are hacked into the code, lack of documentation, documentation getting lost or programmers joining and leaving the team. When adaptations to such eventually complex source code must be made, a great deal of effort is spent on the mere comprehension of the code. Therefore, maintainers can benefit greatly by a better methodology to gain an understanding of the source code. This is what is being researched in the field of Program Comprehension.

Formally, program comprehension is the act of constructing a mental model of a software system. A mental model is “an internal, working representation of the software under consideration” [55] and consists of knowledge about the software system on various levels of detail. For example, on the most detailed level it may contain implementation details such as variable names and their uses, while on the highest level it may contain knowledge about the collaboration between different top level modules.

Program Comprehension can roughly be divided in two different parts: cognitive research, studying how programmers comprehend programs and how information could be presented to better facilitate their process of comprehension (Section 2.1.1); and tool design, putting the theories in practice by creating actual tools that have the potential to help a programmer with gaining understanding of a program (Section 2.1.2).

2.1.1 Cognitive research

Behavioral theorists have long been studying programmers to find out how they comprehend programs, in order to learn how tools should help the programmer. An outstanding review of the past can be read in [15], going back until the 1970s. By now, it is generally agreed upon that programmers incrementally build a mental model of the software system, acquiring knowledge according to one of the following three models:

- **Top-down models.**
  In top-down models, such as the one proposed by Brooks [9], programmers start with a hypothesis about how the program works in general, for which he searches evidence that either supports or refutes that hypothesis. The view in this top level knowledge domain is refined in a hierarchical fashion into more detailed knowledge domains. Thus, a succession of knowledge domains is reconstructed by the programmer, bridging the gap between the problem domain and the executing program. For example, a programmer may start with the assumption that a program is built on a model-view-controller architecture, searching for the classes responsible for the model, the view and the controller part. Further on in the process of understanding, he may hypothesize that an array search operation appearing somewhere in the program uses a binary search, and will look for the typical low, mid and high variables.

  A top-down strategy is generally used by experienced programmers that are already familiar with the programming language and the problem domain, since he is required to formulate an initial hypothesis on how the program might work.
When verifying a hypothesis, *beacons* and *rules of discourse* form an important part of the process [10]. Beacons are operations typical for a certain knowledge domain (such as the above example of the variables in a binary search) and are useful for high-level understanding. Rules of discourse are conventions in programming, such as coding standards or typical uses of data structures, and set programmer expectations. Programmers use this to set hypotheses on the program under investigation.

- **Bottom-up models.**
  In bottom-up models, such as the Pennington model [41], programmers work the other way around, compared to top-down models. They start from the code (the lowest level) and group chunks of statements together into higher level abstractions. This process is called *chunking*. Simultaneously, they identify the purpose of these chunks, a process which is called *concept assignment*. In this way, programmers gradually get a better understanding of the system, and continue with creating abstractions until they have reconstructed the whole program structure, or that part which they need for the current maintenance task.

  Bottom-up strategies are typically used when a programmer is new to the problem domain. Pennington observed that in a bottom-up model, programmers tend to start with examining the control-flow, creating what Pennington calls a *program model* in their mind. In relation to the topic of this thesis, a 2010 study has shown that bottom-up models are very popular when comprehending web applications [35].

- **Mixed models.**
  Shneiderman and Mayer [49], Letovsky [32] and Von Mayrhoaser and Vans [54] all suggested that the way in which a programmer comprehends a software system is not as fixed as the top-down and bottom-up model suggest. Instead, programmers use both strategies interchangeably or simultaneously. All three cognition models are based on the same general idea: the programmer has a *knowledge base*, consisting of what he already knows and a *mental model*, consisting of the current understanding of the software system. Based on an opportunistic process, he assimilates knowledge in either a top-down or a bottom-up fashion, depending on the current comprehension activity.

  Letovsky [32] noted that in all models there are three types of questions that drive the comprehension process of the programmer. These are *why* conjectures (concerning the role of a code block, e.g., “why is this function called in here”), *how* conjectures (how goals are accomplished, e.g., “how is the text made up in a green font”) and *what* conjectures (finding out what a variable is, e.g., “what is variable myVar”).

  Apart from the above ideas, many other considerations have been published. For more in depth information, see the review by Storey [51].

### 2.1.2 Tool design

The second major branch of research within program comprehension is that of tool design. Inspired by both cognitive theories and experience from practice, several ideas
have been proposed and put to the test. Below, several of such ideas will be quickly introduced, to create a frame of reference for the rest of this document.

To facilitate top-down comprehension, documentation is an important aspect. Several tools have therefore been designed to automatically generate this. Some of them require developers to explicitly add meta data to the code, usually in the form of specially formatted comments (such as Javadoc), while others are capable of analyzing the code and generate summaries or diagrams of the structure of the code. Such diagrams, almost always UML diagrams, add a layer of complexity, as large diagrams tend to be very confusing to read. In general, such diagrams can be interpreted as graphs and have gained the interest of the community in the separated branch of research of graph drawing. In this field, an effort is made to draw graphs in a conveniently arranged way and to reduce the amount of redundant information that is presented to the viewer, in order to prevent an information overload. On the other hand, seeing how humans have remarkable perceptual abilities in the visual domain [48], diagrams have been successfully used to implement polymetric views [29], actually increasing the amount of information presented in a view. A polymetric view is a view that employs different types of visualization to present several metrics at once. For example, a class diagram can be extended with metrics such as lines of code or number of methods by giving each class a certain color, dimension or icon.

For bottom-up comprehension, being able to quickly navigate through the code and searching for specific bits of code becomes very important. The engineer typically follows the program flow, which in many modular systems can be scattered across numerous classes or files. Thus, he frequently needs to jump back and forth through the code. Some tools provide automatic navigation (by means of clickable links, e.g., clicking on a variable name takes you to the declaration of that variable), but if that is not present, textual search and query support becomes the method of choice [35].

To keep an overview and not get lost in the jungle of navigation or diagrams, a useful strategy is the use of multiple views, each view showing the software on a different level. Different views can also be useful to highlight different aspects of the system, allowing the engineer to see connections and patterns that would otherwise be left unseen.

As the user gradually understands parts of the system (either using top-down or bottom-up comprehension), he will benefit from support for concept assignment [8]. Tools that offer concept assignment allow the user to link parts of the software to high level tasks. This can for example be achieved by the ability to add notes to selections of the software, or substitution of a group of classes by a conceptual block, named after the feature it implements.

In a completely different approach, researchers have taken an increased interest in the investigation of software evolution [13]. With software evolution, version control systems and other sources (e.g., bug tracking systems or email archives) are mined to automatically identify parts of the software system that may be interesting in the process of gaining a comprehension of the software. For example, a class that is being updated in nearly every revision is likely to be an important class. Thus, it is a good starting point when one reads through the code. Again, polymetric views may be of excellent use when presenting the results to the user.

All in all, visualization is an essential part of tool design for program comprehension. Even as far back as 350 BC, Aristotle stated that “thought is impossible
without an image”, stressing the importance of visualization. An extensive overview of different forms of software visualization that could be used in the field of program comprehension, is given in the book by Diehl [18]. Furthermore, when designing a tool for the visualization of software, one can use the paper by Kienle and Müller [28] to learn from the best practices when building the requirements for that tool, and the survey by Storey [51] can serve as a source of inspiration for picking yet unexplored research directions for new features or tools.

2.2 Web Applications

In the previous section we have discussed some key aspects of program comprehension. Here, we will give a brief overview of the evolution of web applications through time, to understand how the focus of research of the comprehension of web applications changed similarly.

From the dawn of the Internet in 1989 (or 1993 if the date on which the Internet went public is used), it did not take long before web applications made their appearance. The first web applications were sets of static pages, sequentially delivered based upon user input that was given through form elements. Such user input is processed by the server of the application and the right sequence of pages thus creates the interactive experience. In this model, both the client and the server are stateless, which is rarely seen nowadays. However, it made comprehension fairly easy and specific techniques were limited to automated form submission.

In 1994 cookies were introduced, enabling stateful clients. Shortly after, JavaScript (JS) was introduced in 1995, enabling dynamic behavior on the client. However, because of the slow adoption of these techniques at first, it was the server side that changed the most in these days. The increasing popularity of languages like PHP, Ruby, Active Server Pages and Java Server Pages contributed heavily to the development of WAs. Thus, research of the comprehension of WAs was mostly focussed on the server side, while analysis of the client side stayed relatively unchanged.

It must be noted that technologies such as Java applets (1995), Shockwave (1995), Flash (1996) and Silverlight (2007) also enabled dynamic behavior in clients. However, these technologies gave up the classical model of a website, in which a page is represented by a set of HTML elements. Instead of HTML rendered by the browser, the interface was now rendered by the client application in a separate environment. Consequently, existing techniques of website analysis could not be used, or even be adapted to be used, because such analysis tools expected (and still expect) to analyze HTML. Therefore, such technologies will be out of the scope of this thesis.

As stated above, developments on the client side only slowly gained ground. However, in 2005 a turning point was reached in both development and comprehension, when the AJAX technology (Asynchronous JavaScript and XML) was coined [24], marking the most significant change in the paradigm of web applications. This is best stated by Mesbah as follows: “AJAX shatters the metaphor of a web ‘page’ upon which many classic web technologies are based” [36]. The literature presented in Section 2.3 starts only shortly before this new era, since solutions from further back do not account for the dynamic behavior of both clients and servers.
2.3 Solution Methods

Below, thirteen approaches are discussed, taken from literature between 2004 to 2010, focussed on the comprehension of web applications. While reading, note the shift in perception of a web application after 2005, when AJAX technologies largely changed the landscape of web applications.

The WARE approach

In 2004, the results of several years of research by Di Lucca, Fasolino and Tramontana et al. led to the presentation of the WARE approach, short for Web Application Reverse Engineering [17]. Coming from the field of reverse engineering, this approach aims to support the comprehension of existing undocumented web applications to be maintained, through the reconstruction of UML diagrams, providing distinct views of the WA.

The WARE tool uses static and (limited) dynamic analysis of the WA to extract relevant information, which yields a so called Web Application connection Graph (WAG), in which the nodes consist of the entities of the WA, for example web pages or scripts. This graph is structured by the application of a clustering algorithm, whose resulting clusters are validated based on a Concept Assignment Process. From this point, a cluster can be associated with the implementation of a specific (set of) feature(s). Then, with all extracted information, the following UML diagrams are created, according to the Conallen UML model: use case diagrams, class diagrams, component diagrams and sequence diagrams.

The component diagrams represent the implementation view of the WA and are directly retrieved from the results of the static analysis, although the authors do not specify how this diagram is built or what it represents.

The class diagrams represent the structural view of the WA and are retrieved from the results of the static and the dynamic analysis. Each relevant entity is represented as a class, stereotyped with the type of that entry (e.g., a static client page, a form, a server page).

The use case diagrams represent the user requirements view of the WA and are deduced from the clustering results: each cluster may be associated to a use case. The expertise of the reverse engineer is required to decide upon the right set of use cases.

Finally, the sequence diagrams represent the view of the dynamic interactions of the WA. They are derived from the clustered WAG and the dynamic analysis, where the objects in the diagram are the involved classes in that cluster, while the sequence of messages is taken from the dynamic analysis.

WANDA

In 2004, Antoniol et al. [5] proposed to analyze web applications through dynamic analysis and (limited) static analysis. They developed a tool called WANDA, short for Web Applications Dynamic Analyzer, which instruments the code of the application of interest. It does so by nesting itself in the architecture, logging all relevant data when a page is requested. A user then interacts with the WA, producing relevant log data. With this information, three types of diagrams are created: deployment and component diagrams, sequence diagrams and class diagrams.
The deployment and component diagrams show which resources are involved in the WA, i.e., the used web services, databases, files (other than files containing application code), COTS components and clients. The diagrams are enriched with frequency information, showing how many times that resource was used or accessed during the dynamic analysis.

The sequence diagrams show the chronological interaction between the user and pages, and pages and other resources. The entities of a sequence diagram are the users, the HTML pages, the server pages (application code) and the other resources.

The class diagrams are an extension of the Conallen UML model [11], which was developed specifically to model WAs, and are based upon both the static and dynamic analysis. The extra information that is added to the diagrams consists of the type of access (a submit or a link for page navigation; an insert, update or select statement for database access; etc.), the frequency of each access type and the variables passed between pages.

### Integration of WARE and WANDA

With the WARE tool being mostly focussed on static analysis and the WANDA tool being mostly focussed on dynamic analysis, it was no surprise that in 2005 the two tools were integrated [16]. The result was an application with all functionalities of the original tools, plus the ability to identify equivalent client pages. Client pages are considered to be equivalent if they are structurally the same (e.g., they have the same DOM tree, but not the same content), where repetitions of the same structure are also considered the same. The static analysis was used to generate a control flow graph for each page, after which the dynamic analysis was used to identify the paths in this control flow graph that actually yield client pages. Based on these paths, client pages were considered to be equivalent or not, leading to the final set of client pages.

With this ability, the class diagrams, where a client page is represented by a class, were improved in two ways: more unique pages were identified then either tool did by itself, and the diagrams were better understandable then the original ones because several pages were grouped together since they were found to be equivalent.

### Revangie

Instead of looking at the source code of a WA, as was done in the approaches discussed above, Draheim et al. took another approach by only analyzing the client side of the WA [19]. With their tool Revangie they reconstruct a form-oriented model of the WA, consisting of a bipartite graph with a set of pages and a set of actions, called a formchart. A page can consist of multiple webpages, showing instances of the same conceptual page, such as a set of similar webpages, each showing the details of a different item. An action is an action performed by the server in response to a user request, such as the evaluation of a PHP page. In the graph, a page is connected to actions because the page contains hyperlinks to and form elements that are submitted to those actions. Actions are connected to pages, because the response of an action is another page.

This formchart is reconstructed by means of automatic crawling of the web application. Initially, the reverse engineer must specify a set of starting points in the WA,
after which the tool navigates through the application until each actions has, on average, been triggered a minimum number of times. The webpages are clustered together based on a user definable equivalence definition and actions are distinguished from each other based on their URL and the parameters sent with them.

WebAppSleuth

In 2007, Fisher II et al. [20] presented a tool called WebAppSleuth to analyze a form of a web application in a black box setting. More precisely, it finds the relations between the variables that are submitted to the form, without accessing the server side code. It is able to identify all mandatory, optional and mandatorily absent variables, it finds variable implications and it reconstructs value hierarchies.

This is done by repeatedly sending requests with different values for the variables and analyzing the result. A result can be either valid/invalid (gives some piece of information or an error), or it consists of a result set (in which case a non-empty result set is valid and an empty result set is valid). The request that led to a result is classified similarly as the result, e.g., if the result is invalid, the request that led to that result is an invalid request.

A mandatory variable is a variable that is present in all valid requests and absent in at least one invalid request. A optional variable is a variable that is present in at least one valid request and absent in at least one valid request. A mandatorily absent variable is a variable that is absent in all valid requests and present in at least one invalid request.

Three different variable implications are extracted. First, for each variable, the variables that are implied by this variable are found. That is, for each valid request that contains variable \( p \), another set of variables must also be present. Second, a set of variables is extracted, the meaning of which is that these variables must at least be present in each valid request. Third, for each combination of a variable and a value, the variables that are implied by this assignment are found, similarly to the first type of implication.

If the form concerns a search, the variables determine the search results. This allows one to create a value hierarchies for each variable. Keeping all other variables the same, one variable is tried with different values. The results are analyzed and a graph is made indicating which result set is a subset of which other result set. This gives a directed acyclic graph that typically shows a certain pattern, giving more insight into the semantics of the variable.

RE-UWA

Bernardi et al. [7] took a different approach to analyze a WA. They only analyzed the client side of the WA, to reverse engineer user-centered conceptual models. Since their approach is based upon analyzing the html of the front-end, it is applicable to all WAs that have a front-end consisting of html pages. The re-engineered models are diagrams according to the Ubiquitous Web Applications (UWA) design methodology, resulting in the UWA Hyperbase (the elementary objects and the relations between them, e.g., a painting is made by an artist) and the UWA Access Structure models (groups of instances of the objects that are presented to the user, e.g., a group of paintings that is
made by the same artist). This is achieved by crawling the website and analyzing the html of the webpages.

The **UWA Hyperbase** consists of **Entities Types** (similar to classes) and **Semantic Association Types** (similar to class-relations). The entities are found by two methods. In the first method, similar pages are clustered. A cluster is likely to consist of pages showing different instances of the same entity and thus each cluster is a candidate for an entity. The properties (in UWA terminology, *Slots*) of that entity then consist of the similar parts of the pages in that cluster. In the second method, keywords are extracted from forms, tables and semantic definitions on the webpages. The keywords are also grouped, based on where they occurred, giving other candidate entities. The entities are validated manually by an expert. The semantic associations are found by analyzing hyperlinks and pages with occurrences of multiple entities. Again, an expert manually verifies which of these potential semantic associations are valid.

The **UWA Access Structure models** consist of **UWA Collections**, which are selections of entity instances, specifying which views are available to a user to see an overview of some data type. Any page showing a list of instances or hyperlinks to instances of the same entity type gives a candidate collection. Like the entities and semantic associations, these are manually validated by an expert.

The approach has been implemented in a tool called RE-UWA, and experiments have shown that the resulting UWA Hyperbase and Access Structure models are correct. Improvements can mainly be achieved by better keyword extraction by the application of information retrieval techniques.

**DynaRIA**

With the increased complexity of WAs due to the usage of AJAX, the client side no longer remained static. Instead, complex event-based JavaScript functions are executed, asynchronous communication with the server is used and the DOM can change as a result of these JavaScripts. Thus, an important part of the comprehension of WAs becomes to understand the program flow on the client. In 2010, Amalfitano et al. [2] created the tool DynaRIA to facilitate this need, by visualizing the execution of JavaScript code and all associated data in several ways.

DynaRIA was developed primarily for program comprehension activities, but can also be applied for testing or quality assessment. Only the client side of the WA is analyzed, making it independent of the server side. It works by presenting the user with an integrated browser with which the user interacts with the WA. DynaRIA logs all activity, that is to say the fired events, the executed JavaScript, the exceptions and errors that occurred, the DOM changes and the network traffic. This information is visualized in a number of ways, most notably an event sequence, a JavaScript function call-tree, the communication sequence with the server, the *event-flow-graph* (EFG) and a UML sequence diagram. Furthermore, various metrics about the JavaScript are listed, such as execution times and code coverage.

For testing purposes, the captured user sessions can be replayed. During the replay, the tool offers various features for test suite error detection and coverage evaluation. The effectiveness of the test suites can be evaluated using the code coverage metrics. The quality assessment consists of additional metrics, such as JavaScript function size, call coupling between JavaScript modules (the number of function calls between two
modules), server coupling (the number of messages exchanged between a JavaScript module and the server) or DOM coupling (the number of changes made to the DOM by a JavaScript module).

**FireDetective**

Simultaneous with the development of DynaRIA, Matthijssen et al. [35] built a tool for the comprehension of modern WAs using AJAX technology as well, called FireDetective. In contrast to DynaRIA, FireDetective also traces the execution on the server side of the WA. The traces are visualized in a combined way, resulting in a better understanding of the application as a whole.

FireDetective installs a plugin into the browser Firefox and is hooked to the server side. As the user interacts with the WA, traces are recorded and shown live in the visualizer. The traces are detailed to the call level, consisting of functions and methods, and are shown in the order they were called. The traces are linked to abstractions from the AJAX/web domain (such as page requests, AJAX-requests or template invocations) to allow an engineer to answer ‘what’, ‘how’ and ‘why’ questions, such as “why did this AJAX request occur?”.

The user interface consists of four views, showing (1) a tree representation of the AJAX/web domain abstractions, (2) a call tree of the trace selected in the first view, (3) the code from the selected method from the second view and (4) a file browser, showing the resources of the AJAX application. Two mechanisms are applied to increase the comprehensibility of the traces. First, only calls specific to the WA are shown (library calls are omitted). Second, the recording can be started and stopped, allowing to time slice the WA.

**Crawling**

The quintessence of dynamic analysis is the fact that the application under analysis is being execution, and thus being ‘used’. In the previous selection of literature, it was seen that this is usually achieved in one of three ways: (1) the application is inspected while a user uses it, (2) previous uses of the application were recorded, which are now replayed or (3) the application is used by a robot that systematically browses through the entire WA.

This third technique, called crawling, turns out to be useful for more then just the making sure the whole WA is browsed through during dynamic analysis. The RE-UWA tool [7] discussed above sets an example of how information can be extracted from crawling a WA. Another way of using a crawl, is to analyze the resulting user interface flow diagram (alternatively, activity diagram or state flow graph), which shows all possible paths of navigation in a WA. Below, several such proposals will be presented.

Additionally, several clustering methods will be mentioned, since such diagrams typically suffer from the well known state explosion problem [53]. Therefore, several authors have also proposed clustering methods to reduce the size of such graphs,

---

1Currently, the tool is built for WAs with a Java + JSP back-end, but the techniques can be applied to other back-ends as well, such as PHP.
specifically tailored to web applications. More on clustering techniques can be found in [46].

ReWeb

In 2001, Ricca and Tonella presented a tool to analyze the navigational structure of a website, and the evolution of this structure through time [42]. They model a website as a set of pages and frames (which were much more used back then, than nowadays) as nodes, and hyperlinks as edges. Such a navigational graph of the WA can be created using their tool ReWeb, which consists of a crawler, an analyzer and a viewer. No clustering is applied to the resulting graph, leading to incomprehensible graphs, even for relatively small WAs.

Being developed in 2001, it is no surprise that the method does not work for dynamic websites such as those using AJAX, but it does work for the earlier type of web applications, based on forms. In the graph, outgoing edges (hyperlinks) of a page $p$ are annotated with the values of the variables that must be submitted in the request of page $p$ in order for the server to generate that specific hyperlink. Unfortunately, no description is given how these values can be found.

The history is visualized by creating the graph for different instantiations of the website through time (note that, since the tool is not attached to a version control system, this requires that ReWeb is run periodically). These graphs are merged into one, where pages are textually compared to each other to identify modifications to pages. The nodes (pages) are colored based upon their last modification time.

While the particular frame oriented approach is quite outdated by now, the idea of a page flow graph to gain insight into the evolution of a web application is promising.

FSMWeb

In 2005, Andrews et al. proposed to model the behavior of a WA as a hierarchy of finite state machines (FSMs), as part of the FSMWeb method [4]. At the lowest level, an FSM models a single html form or webpage, called a Logical Web Page (LWP). At higher levels, the FSMs are linked to each other to form clusters that represent some functionality of the web application, until at the highest level the agglomeration of the clusters compose the entire WA. The links between the FSMs are annotated with input variables that are used in successive FSMs. With this graph, test cases are generated to test the WA.

The graph described above can be used by engineers to get insight into a WA, by being able to see how the WA is used by its users. The graph is created by analyzing the WA in a black box setting, using only the webpages from the client side, which can be done by crawling the WA and identifying all LWPs. FSMs are extracted automatically, but the clustering is done manually. The annotations are also defined manually.

Similarly to the previous method (ReWeb), this method is not applicable to modern web applications, since the dynamic nature of webpages is not yet taken into account. The form based approach is better then the frame based approach of ReWeb, but still insufficient, because any component can trigger events in a modern WA, instead of only submit buttons and anchor elements.
2. BACKGROUND AND RELATED WORK

Figure 2.1: The conceptual model of the behavior of the client side of a web application, used by RE-RIA [3]

RE-RIA / Creria

In 2008, Amalfitano et al. proposed a novel way to reverse engineer an FSM, modeling the client side behavior of a web application by means of dynamic analysis [3]. Just like FSMWeb, they create an FSM for the application, but they assume a much more flexible model of a WA (see Figure 2.1), accounting for all possible behavior of modern WAs. They reason that since the user interface of a WA is defined by the DOM of the webpages, the instantiations of the DOM should be used as the states in the FSM. Thus, not only does each webpage become a state, but also all ‘derived pages’—due to changes in the DOM—become states. Events that cause a change in the DOM (e.g., the clicking of a link or the handling of a response of an AJAX request) form the state transitions. A tool called RE-RIA (later renamed to Creria) was built and evaluated to proof the concept. Interestingly, FSMExtractor and Crawljax, which were developed independently from RE-RIA, use a similar model of a web application, also accounting for DOM changes without page reloads and acknowledging the fact that any element can cause a state transition.

The FSM is constructed in three steps. First, the user navigates through the application while the tool records all events and the DOM between those events. This forms the basis of the FSM, called a transition graph. Then, based on equivalence criteria, states and transitions are clustered together. For example, two states may be equivalent if the DOM structure is the same, and two transitions may be equivalent if they were caused by the same event type and are associated with the same DOM element and event handler. Third, the engineer manually validates and, if needed, adapts the clusters and simultaneously performs concept assignment on the resulting clusters. This results in a potentially compact FSM, depending on the quality of the equivalence criteria and the engineer.
2.3. Solution Methods

FSMEExtractor

A similar method was applied by Marchetto et al. in the same year, building an FSM of a WA, ultimately to derive test cases from it [34]. The FSM produced by their FSMExtractor has states that represent DOM abstractions and state transitions that correspond to methods changing the DOM abstraction. A DOM abstraction is a structural description of the DOM, abstracting the actual content that is shown in that DOM instance. After the WA has been traced, a manual verification step is required to add/remove states and/or transitions that were not accounted for during the dynamic analysis or resulted from faulty traces.

The traces are made by tracing the set of methods that potentially change the DOM. Static analysis is used to derive this set of methods. These methods are then traced while the user executes several scenarios with the application. With the information gathered in this way, the tool reconstructs the FSM, using a user-defined abstraction function that maps a DOM instance to a DOM abstraction. Although no clustering step is applied in this process, the size of the FSM may still be limited if the DOM abstraction function is able to get rid of all redundant content, effectively implementing a similar feature as clustering based on equivalence criteria.

Crawljax

Simultaneously with RE-RIA and FSMExtractor, the tool Crawljax was released in 2008. Crawljax features a customizable automated crawler for web applications that constructs a state-flow graph (SFG) of the entire WA [36]. The states in this graph represent DOM instances and the state transitions represent the elements and/or events that caused the transition. Since Crawljax systematically crawls the entire application, the SFG corresponds to the complete navigational graph of the WA. It can be used for several purposes, such as indexing of the WA, various forms of testing of the WA or comprehending the WA.

To create the graph, Crawljax repeatedly clicks on elements that will potentially modify the DOM, compares the DOM before and after the click and records a new state (or loads a previously recorded state if it is already visited) if the DOM is changed more than a predefined threshold. It continues in a depth-first manner until all candidate elements in all states are investigated. This process is customizable in several ways, making it possible to enhance both the quality and the speed of the crawl.

First of all, the candidate elements are by default chosen very liberal (selecting rather way to many then a little to few), minimizing the chance of missing a link. However, with application specific knowledge, it is possible to reduce the set considerably. Second, the threshold that determines if a new state is entered can be altered. Third, in recent versions of Crawljax a plugin API has been added, making it is possible to define a variety of custom features to the crawling process, such as a custom equivalence function to determine if two states should be considered equal. This also opens the door for attaching basically any feature to Crawljax, such as executing test suites [38], exposing a static version of the WA to search engines or visualization for comprehension purposes.
2. BACKGROUND AND RELATED WORK

2.4 Available Tools

Section 2.3 was largely focussed around the academic side of the comprehension of web applications. While most results from literature were accompanied by a proof of concept, it turns out that this does not always guarantee that the proposed method or approach can be easily applied in practice. Carlo Ghezzi remarked in 2009 that of all the papers that dealt directly or indirectly with tools and appeared in the ACM Transactions on Software Engineering and Methodology between 2000 and 2008, only 20% of those were actually installable, let alone functional [14]. This shows that, in order to get a usable overview of the current state of the art, it does not suffice to merely review the suggested methodologies in literature alone, but to explicitly look at the available tools. Another reason to do so is that not all tools come from the academic world, but are commercial off-the-shelf (COTS) tools. This section will be devoted to a quick summary of the most popular or promising tools, both COTS, academic and any other form available.

The list of applications shown below were selected for their capability to contribute to the understanding of a web application. As a deliberate choice, debuggers were left out of this list, since they are not designed with program comprehension in mind, but for the mere ability to hunt down bugs. However, one debugger was selected for its ability to show function traces and code coverage.

Selected tools

The tools that will be discussed in this section include all tools that were presented in the previous section, plus six tools that did not originate from the academic world, but instead were found within the online community. These are phpCallGraph\(^2\), Xdebug\(^3\), Doxygen\(^4\), phpDocumentor\(^5\), Firebug\(^6\) and dynaTrace AJAX Edition\(^7\). Several other tools were found as well, but are not selected because they do not include program comprehension features. Instead, they were focussed on other aspects such as checking coding standards (PHP_CodeSniffer), source code validation (PHP-sat, PHPLint), code metrics generation (PHP Depend), profiling (XHProf) or security auditing (Rational AppScan). Many other tools are no doubt still available on the Internet (e.g., FireCrystal and Script InSight), but in the interest of time it goes beyond the scope of this report to extend the list any further. Thus, the list of tools to look into is as follows:

- WARE
- WANDA
- WARE + WANDA
- Revangie
- WebAppSleuth
- RE-UWA
- DynaRIA
- FireDetective
- ReWeb
- FSMWeb
- RE-RIA / Creria
- FSMExtractor
- Crawljax
- PHP Call Graph
- xdebug
- Doxygen
- phpDocumentor
- FireBug
- dynaTrace AJAX Edition

\(^2\)http://phpcallgraph.sourceforge.net/
\(^3\)http://www.xdebug.org/
\(^4\)http://www.doxygen.org/
\(^5\)http://www.phpdoc.org/
\(^6\)http://getfirebug.com/
\(^7\)http://ajax.dynatrace.com/ajax/
<table>
<thead>
<tr>
<th>Name</th>
<th>Last activity</th>
<th>Primary focus</th>
<th>Program comprehension facilities</th>
<th>Side</th>
<th>Type</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Revangie</td>
<td>2005</td>
<td>WA comprehension</td>
<td>Constructs a form-chart showing which client page is connected to which server page. Does not evaluate JS.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>WebAppSleuth</td>
<td>2007</td>
<td>WA comprehension</td>
<td>Thoroughly performs automatic analysis of forms. Deals with single forms only, so does not need to evaluate JS.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Automatic</td>
</tr>
<tr>
<td>phpDocumentor</td>
<td>2008</td>
<td>PHP documentation</td>
<td>Can generate class inheritance diagrams for PHP code.</td>
<td>Server</td>
<td>Static</td>
<td>N/A</td>
</tr>
<tr>
<td>phpCallGraph</td>
<td>2009</td>
<td>PHP comprehension</td>
<td>Generates the call graph of PHP functions calling each other. Groups classes together.</td>
<td>Server</td>
<td>Static</td>
<td>N/A</td>
</tr>
<tr>
<td>DynaRIA</td>
<td>2010</td>
<td>WA comprehension</td>
<td>Visualizes the event sequence, the JS function call-tree, the event-flow-graph and communication with the server.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>RE-RIA/Ceria</td>
<td>2010</td>
<td>WA comprehension</td>
<td>Generates an FSM with DOM instances as states. States and transitions are clustered based on fixed equivalence criteria.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Crawjax</td>
<td>2010</td>
<td>WA testing</td>
<td>Generates a state-flow graph of the WA, but does not visualize it. Provides a flexible plugin mechanism.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>FireDetective</td>
<td>2011</td>
<td>WA comprehension</td>
<td>Visualizes execution traces, mixing those from the client and the server.</td>
<td>Client &amp; server</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Xdebug</td>
<td>2011</td>
<td>PHP debugging</td>
<td>Can generate function traces, logging each entry and exit, together with parameters and return values.</td>
<td>Server</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
<tr>
<td>Doxygen</td>
<td>2011</td>
<td>PHP documentation</td>
<td>Can generate several inheritance and dependency graphs for PHP code.</td>
<td>Server</td>
<td>Static</td>
<td>N/A</td>
</tr>
<tr>
<td>Firebug</td>
<td>2011</td>
<td>WA debugging</td>
<td>Provides features to easily access and browse through the HTML, JS, DOM and CSS of a WA on the client.</td>
<td>Client</td>
<td>Dynamic</td>
<td>Manual</td>
</tr>
</tbody>
</table>

Table 2.1: A summarizing list of tools that can be used for the comprehension of web applications. The column labeled ‘Last activity’ shows the year of the last release of the program (measured in 2011), the column ‘Primary focus’ describes the main goal of the tool, ‘Program comprehension facilities’ summarize the usefulness to this survey, ‘Side’ indicates if it is run on the client, server or both (‘client only’ hints black-box analysis), ‘Type’ indicates if the analysis is static, dynamic or both and ‘Control’ indicates if the dynamic analysis (when applicable) happens during user interaction or a robot browsing through the WA.
From this list, the tools that, to the best of my knowledge, could not be downloaded were left out. These were WANDA, the combination of WARE and WANDA, RE-UWA, ReWeb, FSMWeb and the tool set from which FSMExtractor is part of. Furthermore WARE was left out, because it has not been updated since 2002 and still relies on legacy software such as Microsoft Access 97.

With this list, the overview from Table 2.1 is compiled which enables one to easily compare the different solutions to each other, so one can make an educated decision as to which tool one has to use in order to accomplish some goal. The overview includes the last year of activity, the primary focus, the program comprehension facilities, the side of the WA that is being analyzed (client and/or server side), the type of analysis (dynamic and/or static analysis) and, in case of dynamic analysis, how the application is controlled during the analysis (the user interacts with the WA: manual, or the tool performs some sort of automatic crawling: automatic). Below follow some explanatory notes accompanying Table 2.1.

The most important column is the one labeled ‘Program comprehension facilities’. It describes why the tool may be interesting for the comprehension of web applications and thus why it has been selected. The observant reader may have noticed a large amount of references to JavaScript and PHP. As for the latter, some tools were searched and found deliberately in the PHP community, since PHP is a frequent language of choice for web applications. Thus, as a natural result this list is biased towards PHP tools. As for the former, let us consider the statement by Mesbah et al. [36] that “crawling AJAX is fundamentally more difficult than crawling classical multi-page web applications”. Since AJAX is implemented in JavaScript, the fact that a tool is able to understand and deal with JS is important if the WA to analyze is (partly) built upon AJAX technology.

The last year of activity is included to get a crude indication whether or not the application is still in development, which usually means that new features are added regularly, making them potentially more useful then similar applications not under development. The primary focus column is added to see what kind of new features are most likely to be expected in new versions of the tool. The side that is being analyzed basically tells whether or not the tool has to have access to the source code, or has to be integrated into the source, in which case it would be a server side tool. Client side analysis tools can usually be seamlessly attached to the live application. The type of analysis shows whether the tool uses static or dynamic analysis, or both. The control column is only used for tools that employ dynamic analysis and is added to indicate how thorough the analysis is. If the control is manual, it depends on the engineer what parts of the WA are analyzed, which can be beneficial if one wants to focus the analysis on a specific part of the WA. On the other hand, if one is interested in an analysis of the entire application, one should pick a tool that is controlled automatically, giving a systematical rather then an intuitive result.

Two interesting things can be seen in the table. First of all, tools that analyze the client side never apply static analysis. This can be explained by the fact that the client side code (the HTML and scripting code) is almost always generated dynamically by the server. Thus, to analyze the client side in a static context would mean to thoroughly analyze and interpret the server side code. That would be far more complex then just running the application and see what the server returns.

Second, static and dynamic analysis is never both applied by one tool. This is inter-
2.4. Available Tools

esting, because several authors [5, 2], both further in the past and recently, have stated that “static code analysis is not sufficient”. Although this does not necessarily mean that static analysis has to be combined with dynamic analysis, it is being insinuated. To the best of my knowledge, three projects have been proposed that apply both static and dynamic analysis: WARE, WANDA and the joint effort of the two [17, 5, 16]. All of these tools were filtered from the above listed selection, because the tools were either too old or not downloadable at all, showing that they did not survive the ravages of time. One can draw either of two conclusions from this observation: combined with the success of several tools that rely solely on dynamic analysis, the authors were wrong in suggesting that static and dynamic analysis have to be combined to get satisfactory results (instead, dynamic analysis suffices); or, there is a branch of research left uninvestigated.
Chapter 3

Approach

Within the field of program comprehension, there is an interest in the visualization of applications. We feel that within this interest, the branch that is tailored towards web applications is worthy of further exploration because web applications are so ubiquitous nowadays [37] and relatively few work exists within this branch [12]. One way to visualize an application is by generating and showing the state-flow graph. For web applications, this boils down to a graph consisting of all unique pages that the web application contains or generates (the nodes) and the possibilities of navigation between these pages (the edges).

Several tools already generate or show the state-flow graph of a web application (see Section 2.4), but these graphs are all static in nature. Because it is well known that programmers often use different comprehension strategies to achieve different tasks [52], it is our believe that the comprehension of web applications can be improved by adding dynamic behavior to these state-flow graph visualizations. We intend to do this by including an adjustable layout, enriching the graph with state based meta data and finishing this with intuitive navigation.

Our next focus derives from the interface between automated testing and program comprehension, specifically the interpretation of the results of automated test suites. Traditionally, when a tester writes his tests, he is fully aware of what parts of the program are tested by which tests, and these tests contribute to a developers understanding of the program [39]. However, this awareness is hard to maintain when the tests are automatically generated [43]. By incorporating the test results of an automatically generated test suite into a visualization of the application, the mapping between a part of the program and the tests conducted there is directly visible.

This chapter will start with describing the concepts that we propose to implement in Section 3.1, which follow from the observations mentioned above. Next, the tool that we built is described in Section 3.2 and we end with a short summary of the approach taken in this thesis in Section 3.3.

3.1 Proposed concepts

As stated above, the two main concepts of this thesis are 1) the interactive visualization of state-flow graphs of web applications; and 2) the integration of test results into a visualization of the application under test. For practical purposes, the visualization
chosen for the test results in 2) is the state-flow graph from 1), enabling us to merge
the two concepts in one tool, fitting all our work in a single look-and-feel. This makes
the visualization a polymetric one [29], showing both the structure of the application
and the test results. While we believe that the power of polymetric views is not yet
fully employed with merely these two concepts, it is outside the scope of this project
to add more functionality. However, while building the proof of concept for the two
previously mentioned concepts, we have added hooks and interfaces to let everyone
add more views and metrics into the visualization.

3.1.1 Graph layout algorithm

For the visualization, first of all a suitable graph layout algorithm must be chosen.
There exist many layout algorithms and a good survey on this topic can be found in
[26]. The properties that are desirable in the dynamic visualization that we propose
are the following:

- The graph layout should reflect the structure of the web application.
- If a clear division can be made between parts in the web application that are
  relatively well separated, this should be reflected in the graph layout.
- States that are more similar to each other than other states should be positioned
  closer to each other than those other states.
- To optimally use the screen real estate, each state should be allocated more or
  less the same amount of space.
- Users should be able to make adjustments to the layout to match their current
  intents.

Two choices that we think will match these constraints are a hierarchical layout
[56] and a force-directed layout [22]. The hierarchical layout can be a good choice
since many web applications have a hierarchical structure and the layout would there-
fore adequately reflect the structure of the web application. However, web applications
also usually have many internal links that would show as cross-links spanning the en-
tire hierarchy (which would look like Figure 3.1). To avoid this clutter, we propose
to use the force-directed layout. We expect that if a clear hierarchy is present in the
state-flow graph, a force-directed layout will group the nodes from each sub hierar-
chy together just like a hierarchical layout, yet finds a more suitable placement for the
nodes whenever the presence of many internal links start to obscure the structure, as
observed by Gansner [23]. Furthermore, in a force-directed layout the whole graph
can simply be reordered by dragging a single node and updating all nodes according
to the new forces, enabling users to customize the layout without losing its favorable
properties. This layout will be the starting point of the interactive visualization and
acts as the overview of the entire web application.

3.1.2 Clustering

Because even medium sized web applications tend to yield large state-flow graphs,
an overview of the entire system quickly becomes uncomprehensible. To increase the
comprehensibility of such an exploded diagram, the number of shown artifacts (states,
edges, labels, etc.) can be lessened by clustering nodes together. This gives a less
3.1. Proposed concepts

Figure 3.1: A hierarchical layout with many cross-links, illustrating the clutter that can arise when using this layout

cluttered graph and is therefore easier to understand. By creating a hierarchical clustering\(^1\) [47] of the nodes in the state-flow graph, where high level clusters represent a high level of abstraction of the structure of the web application and low level clusters represent the fine grained details of the structure, the user can be presented with different levels of detail. Depending on the comprehension strategy that he then currently uses, he can focus on different aspects of the structure. For a crude overview of the application, the top level of the cluster hierarchy is very suitable. For focusing on a single part of the web application, repeatedly expanding more fine grained clusters can achieve the users goal.

3.1.3 Edge filtering

Despite the application of clustering and a suitable layout algorithm, there is still the possibility that a web application yields a state-flow graph that is difficult to interpret because an abundance of edges clouds the structure we are looking for, even with the application of the right layout and clustering. For example, consider a web application with a simple menu structure with \(n\) menu items in it that is present at every page. Because the menu is present at every page, every node in the graph will have \(n\) outgoing edges for the menu items. On top of that, the \(n\) landing pages to which the menu items link will have an incoming edge from every other node. All these edges do not contribute to a better understanding of the application’s structure but only cloud the image with a plethora of edges. Therefore, we propose that the visualization should make use of a filtering mechanism, leaving out edges that are regarded as superfluous.

Closely related to such superfluous edges are edges that contribute less to the comprehensibility of the structure than others, but should not be filtered out. Such edges can be drawn in a less noticeable color than other edges, for example a shade of gray, drawing the attention to other edges. It is hard to define which edges are less important than others and therefore we will base this decision on the following observations.

\(^1\)Not to be confused with the hierarchical layout discussed earlier, however, presumably with a similar hierarchy.
3. Approach

Because visual clutter will always have a negative impact on the comprehension, and because visual clutter is more likely to occur around nodes with high degrees, we propose that the shading is a function of the in- and out-degree of the nodes connected by the edge.

3.1.4 Interactive navigation

As stated before, a key aspect of these proposals is that they should be interactive. For the graph layout this means that the user must have the ability to rearrange nodes as he likes. In order to keep the forces in a force-directed solution meaningful, it can be preferred that if one node is dragged, all nodes close to this node will be dragged along, as if they were attached by rubber bands. This behavior will be included in the proof of concept.

As for clustering, the interactivity will consist of the ability to expand and collapse clusters, which introduces a challenge for the graph layout, known as preservation of the mental map [6]. When a cluster is expanded, several previously hidden artifacts (states or clusters) need to be shown. Due to the force-directed placement, this influences the position of other artifacts as well, possibly changing the layout so significantly that it destroys the user’s mental map. To lessen this problem a mental map preserving technique will be used that gradually morphs the graph layout into a new layout. Experiments by Archambault and Purchase have shown that this is indeed an effective measure [6].

Lastly, interaction will be accomplished by showing a separated view with fine grained details of the currently selected artifact in the graph. In case of a selected state, this includes for example an image depicting how the web application looked in that state, the URL and all possibilities of navigating away from that state. Aside from the system wide overview in the form of a graph, this presents the user with a way to zoom in on the system.

3.1.5 Integrated test reporting

Automatically generated test suites are often hard to understand [43] and consequently, the results can be hard to interpret. To help the developer better understand the test results, we intend to integrate those results into the visualization. For the purpose of our proof of concept, we focus upon automatically generated regression tests for web applications as described in [38].

First of all, we will help the developer with quickly determining the most interesting tests to investigate: the failed tests. Since tests are executed per state, we can add a visual cue on each state in the state-flow graph to indicate the number of failed tests in that state. Within the detailed view of a state, the user will be able to see all tests executed in that state, clearly marking all failed tests. Next, we will help the developer to understand what was tested by which test by linking each test to the actual artifact that was tested within that state, as far as no ambiguity arises.

The same concepts are applied to clusters. Since clusters are merely a collection of states and/or other clusters, the results of the states contained by a cluster can be aggregated for that cluster. Within the visualization, a cluster can then be provided with the same visual cue as was added to a single state. Within the detailed view of
3.2 Graphicrawl tool

To evaluate the concepts proposed above, we have created a tool that implements them, Graphicrawl. This section will start with a description of the environment in which the tool is developed, e.g., the framework upon which it was built (3.2.1). Based upon the environment, the architecture of Graphicrawl was created. The architecture gives an overview of the different components and how they implement the concepts proposed in the previous section (3.2.2). Finally, a thorough description of the user interface of the tool is given (3.2.3).

3.2.1 Environment

Graphicrawl is built upon existing solutions, consisting of a selection of previously made frameworks that implement important features needed as the input for Graphicrawl. These solutions determine the environment in which Graphicrawl operates.

The most important of these is Crawljax [38], a framework that is capable of crawling a website and generating its state-flow graph. Crawljax was chosen because, at the time this research started, it was, to the best of our knowledge, the only crawler available that was capable of crawling websites whose states rely heavily on JavaScript. Other crawlers only had the ability to follow links that are plainly present in the HTML of a website, not accounting for all state transitions that are caused by the execution of dynamic client side code. Since we target web applications, and web applications are by definition reliant upon JavaScript, Crawljax was the logical choice to make.

Automatic regression testing is the second most important part of the surrounding environment of Graphicrawl. The framework we chose to generate and execute the test suite is a plugin developed for Crawljax in 2010 [45]. This plugin is capable of generating invariants for the JavaScript and the Document Object Model (DOM) of a web application. For the purposes of the proof of concept, we decided it was sufficient to incorporate only one of these test suites into the visualization, in this case the automatic regression testing of the DOM.

Finally, for the visualization of the state-flow graph we chose the Jung framework\(^2\), because it provides built-in support for several of our intents for the visualization and has a pluggable interface for all aspects that it does not support out of the box.

3.2.2 Architecture

Based on the environment in which Graphicrawl is situated, we developed the architecture for Graphicrawl. The model applies a typical Model-View-Controller (MVC) pattern and is presented in Figure 3.2. The model contains the data and raw results, the controller directs the process of data generation and interpretation and the view presents the processed data to the user.

\(^2\)http://jung.sourceforge.net/
Within the figure, one can see how we implemented the concepts mentioned in the previous section. The ‘Graph View’ is mainly concerned with the proposed graph layout (3.1.1), but it should come as no surprise that it is involved with the implementation of all concepts, given that all concepts are integrated into the graph visualization. That being said, the clustering (3.1.2) is implemented by the ‘Clusterer’, an ‘Edge Filter’ can be added that decides which edges are filtered out and which edges are more important than other edges (3.1.3) and the integrated test reporting (3.1.5) is handled by the ‘Test result processor’ and the ‘Test Result View’. The interactive navigation (3.1.4) then is a result of seamless interaction between all components.

During the execution of Crawljax, Graphicrawl collects the data used in the visualizations that follow. These data are then stored persistently so the results can be reviewed at any time without the need to recrawl the entire website, a process that we found can take up to half an hour even for a small web site. If one follows the connections between the different components in the architecture, one will notice that eventually everything ends up in the ‘States’ component. This is a natural result of the central role that a state plays in Graphicrawl: all data are linked to a state, the state that the web application was in when that particular piece of data was generated.

### 3.2.3 User interface

The user interface of Graphicrawl can be divided into three parts: the Graph View, the Details View and the Test Result View. This can be extended with customized views,
3.2. Graphicrawl tool

Figure 3.3: The user interface of Graphicrawl, showing the Graph View and the Details View

for example if someone has created a new plugin for Crawljax that demonstrates a novel way of automated testing, and he wishes to visualize these results.

Figure 3.3 shows the user interface of Graphicrawl after crawljax has finished and was instructed to run the regression test suite generated by the test plugin. The left half of the application shows the visualization of the state-flow graph, the right half shows the Details View, here showing a screenshot of the selected state of the example web application that was crawled. An example of the Test Result View can be seen in Figure 3.4, discussed below.

Graph View

In the Graph View (the left half of Figure 3.3 and 3.4) a distinction can be made between clusters and nodes. The rectangular items are the nodes, showing a little screenshot (thumbnail) of the state that it represents. The circular items are the clusters, showing a stacked view of at most three states within that cluster. Deciding which thumbnails are depicted in the stacked view of a cluster should not matter much: if the states are indeed quite similar to each other (i.e., if the clustering is done well), so will be their thumbnail. The figures also clearly show the result of a force-directed layout and the edge filtering. In this particular case, the layout has made sure that all nodes are positioned in such a way that none of the edges cross each other and the edge filtering prevented the graph from becoming an almost complete graph. All these measures should make the graph easier to comprehend, allowing the user to focus on the bare structure instead of getting lost in a crisscross of lines.

The color overlay is determined by the test results. A green overlay means that all invariants checked within that state or cluster passed, no color overlay means that
there were no invariants and a red overlay signals the presence of failed invariants. The rate of transparency is an indication of the number of failed invariants in that node: if 100% of the invariants failed, it will show in bright red, while a percentage close to zero produces only a light shade of red. Thus, one can determine that in the shown example only a few failed invariants are found.

Details View

The Details View (the right half of Figure 3.3) is populated based on the current selection in the graph. In the example a state was selected, resulting in a full sized screenshot being shown in the Details View. Within the screenshot, all elements that triggered a state transition while crawling are indicated by placing a border around it. When the user hovers over such an element, a preview of the state to which this transition leads is shown next to it (as seen in Figure 3.3). Clicking that element will simulate the state transition by selecting the state to which the transition leads. This works vica versa when the user hovers over a state or a cluster in the Graph View: the preview will be shown in the Details View and clicking it will follow the appropriate state transition. The position of the preview in the Details View depends on whether the element that triggered the state transition in question is visible or not. This should allow the user to intuitively and interactively navigate through the state-flow graph, providing multiple ways to achieve the same goal so the user can use the method that is most intuitive for him.

If a cluster is selected, the Details View will render a grid of small previews of all states within that cluster, providing a thorough overview of the contents of that cluster. Similarly to the view of a single state, previews are shown next to the small states when hovered over, and clicking upon a state again navigates the user into that state. This overview should help the user with understanding which part of the application this cluster represents by leveraging the power of the human brain to quickly record repeated patterns in the small previews and recognizing them as patterns they know to exist in the web application.

Test Result View

If there are test results present in the selected state or cluster, they will be shown in the Test Result View (the right half of Figure 3.4). In the top of that view the invariants are listed, while the DOM of the selected state is shown at the bottom of the view. The plugin used to generate the tests (Roest [45], read more in Section 4.3) generates a hierarchical set of invariants, each describing a DOM element that must be present in the DOM of the tested state of the web application. On top of that, the hierarchy and order of the tests must be the same as in the DOM. This hierarchical nature is also shown in the listing of the test results. Similar to the colors used in the Graph View, passed invariants have a green background while failed invariants are shown in red.

When the user selects one of the passed invariants, the corresponding part of the DOM will be highlighted. The corresponding part consists of the element that caused the invariant to pass. In this way, the user can easily match the generated invariants with the DOM of the tested web application. If there is an ambiguous case, i.e., if multiple elements satisfied the invariant, all possible matches will be highlighted.
3.2. Graphicrawl tool

Figure 3.4: The user interface of Graphicrawl, showing the Graph View and the Test Result View (with only passed invariants)

Figure 3.5: The user interface of Graphicrawl, showing the Graph View and the Test Result View, with failed invariants
Figure 3.5 shows an example in which a cluster that contains failures is selected. An unfortunate consequence of a failed invariant is that nothing can be highlighted, since no matching element was found. In order to figure out what went wrong, the user can select the passed invariants immediately preceding and following the failed invariants. This enables the user to view the DOM around the place where the test failed, enabling him to investigate the error.

3.3 Summary

This chapter described the approach that was taken in this thesis in order to improve the comprehension of web applications, by enriching the static visualization of the state-flow graph of a web application and by adding support to locate errors. We have introduced a tool called Graphicrawl that shows an interactive visualization of the state-flow graph aimed at comprehending the structure of the front end of a web application, combined with the test results of automatically generated and executed test suites. To further improve the understanding of the graph, we have applied clustering and edge filtering to reduce the clutter that is inherent in large graphs. To support the locating of errors, the tests from the test suites are matched to the state of the web application in which they were tested and to the corresponding part of the DOM of that state.

For future reference we conclude with a list of the features that were discussed above:

- An interactive visualization of the state-flow graph in the Graph View.
- Both automatic and manual layouting of the state-flow graph in the Graph View.
- The display of full size image captures of the states in the Details View.
- The display of previews in the Details View when hovering over the states in the Graph View or the Details View.
- Navigating through the state-flow graph via both the Graph View and the Details View.
- Application of edge filtering and shading in the Graph View.
- Application of a hierarchical clustering of the state-flow graph.
- An overview of the states in a cluster in the Details View.
- An overview of the invariant hierarchy of the generated regression tests in the Test Result View.
- Establishing a direct link between an invariant and the corresponding HTML in the Test Result View.
- Filtering the invariant hierarchy to isolate failed invariants in the Test Result View.
Chapter 4

Implementation

The proposals described in Chapter 3 have been implemented in a tool called Graphicrawl. In this chapter, we describe the implementation of the main features of this tool: the model, construction and visualization of the state-flow graph (Section 4.1); the clustering algorithm (Section 4.2); the model, construction and visualization of test results (Section 4.3); and all extension points in Graphicrawl where future contributors can modify or extend the functionality of the respective components (Section 4.4), as far as they have not yet been described previously in this chapter.

4.1 Graph architecture

The graph visualization is the most important and central part of the GUI and, consequently, the architecture supporting these graphics is spread out over the entire application, each part fulfilling a single job. Figure 4.1 shows an abstraction of the architecture related to the graph visualization.

4.1.1 Graph construction

First of all, the state-flow graph must be generated. This is done by the Crawler from Crawljax, which uses the domain independent framework JGraphT to represent the mathematical structure of the graph. The objects that represent the nodes in this graph are the States (unique pages in the web application) and the edges in the graph are represented by Events (any click on a page that leads to a state change). This graph is wrapped by Crawljax by a domain dependent wrapper class, the StateFlowGraph, adding several convenience methods for the addition of new states and state transitions, and for performing some basic mathematical operations on the graph such as calculating the shortest path from A to B.

During the crawling phase, the GraphicrawlPlugin is registered in Crawljax so it can record data associated with the states and state transitions in the state-flow graph. This data is stored in Graphicrawl’s AnnotatedSFG, which is a wrapper class.

---

1Note that this is an abstraction. Some components of lesser importance are left out and some relations are more complex in reality than shown here.

2http://www.jgrapht.org/

3Actually called StateVertix’s, but for simplicity here called States.

4Actually called Eventables, but for simplicity here called Events.
4. IMPLEMENTATION

Figure 4.1: The graph architecture

for Crawljax’s StateFlowGraph, and is used by the visualization to determine its appearance. The annotations that are stored in the AnnotatedSFG are the following:

- The position and size of all elements of the web page that triggered a state transition when clicked upon
- All tested invariants, grouped per state
- All failed invariants, grouped per state
- Several caches, such as one that holds the percentage of failed invariants in a state
- Any user defined annotation

With the ability to add user defined annotations, future contributors can add their own custom annotations to this class, allowing them to take advantage of Graphicrawl’s functionality while adding their own. A custom annotation can be any Serializable class, allowing full flexibility. These annotations can subsequently be used after the crawling phase to display metrics in a custom view (see Section 4.4.2).

When the crawling phase is completed, a set of post processors are executed, among which is the clustering algorithm (see Section 4.2). The resulting clustering is again stored in the AnnotatedSFG. Users of Graphicrawl can add their own post processors, if necessary, to process for example the annotations they added themselves.

4.1.2 Graph visualization

The visualization of the graph is made possible through the interaction between the GraphManager and the Graph View. The GraphManager knows about the underlying

---

5Actually called GraphPanel, but for consistency with Figure 3.2 here called the Graph View.
4.2. Clustering algorithm

In order to cope with large scale graphs, we have created a clustering algorithm that groups similar states in the state-flow graph together. This is done in a hierarchical fashion, where both clusters and states can be grouped together in new, coarser clusters. Such a clustering scheme is known as a hierarchical agglomerative clustering algorithm, first described as such by Johnson [27], here described in Section 4.2.1. We shall propose an extension to this scheme in Section 4.2.2 and discuss four different dissimilarity measures used in these algorithms in Section 4.2.3.

4.2.1 Agglomerative clustering

In an agglomerative clustering scheme, a series of agglomeration steps is performed to agglomerate states and/or clusters together into larger clusters, until all states are grouped together in one root cluster. To decide which states or clusters need to be grouped together, a dissimilarity measure must be devised which, given two states, decides how different they are. A value of 0 means the two states are identical, and increasing values indicate a decreasing relation. The selection of a dissimilarity measure is discussed in Section 4.2.3.

Next, a linkage criterion must be defined to determine when a cluster can join with another cluster or state. Two well known linkage criteria are for example complete

---

http://jung.sourceforge.net/
4. IMPLEMENTATION

linkage, where two clusters are joined when all pairs of states between those clusters have a dissimilarity below a threshold, and single linkage, where only one pair of states between two clusters needs to have a dissimilarity below a threshold in order to join those two clusters. We chose to use complete linkage, as this guarantees the most pairwise cohesion within each cluster. In Section 4.4.1 is described how a different linkage method can be used.

When applying an algorithm that implements this scheme, one will end up with a hierarchy in which each cluster exists of exactly two subclusters or states, i.e., a binary tree. While this works in many other contexts, in other contexts it can be desirable to have clusters with three or more immediate children. In our context, this situation might arise when there is a low dissimilarity between each pair of states in a set of three or more states. After application of the agglomerative clustering algorithm, the states from this set will be descendants of one cluster, but not immediate children, since a binary tree only has two children per node. Therefore, we devised an extension to this scheme, called multi agglomerative clustering, which adjusts the binary tree to a regular tree by finding such situations as described before and reordering the tree by making those states siblings of each other.

4.2.2 Multi agglomerative clustering

Within multi agglomerative clustering, we aim to cluster states together in a hierarchy of clusters, where each cluster contains two or more smaller clusters or states. To achieve this, we first apply a normal agglomerative clustering algorithm and then find clustering steps in the hierarchy that could have been omitted. Such steps can be found by looking at clusters for which the pairwise dissimilarities between its siblings and children are below a given threshold. Because this threshold needs to be increased for coarser clusters, it has been defined not as a fixed threshold, but as the difference between the minimum and the maximum dissimilarity of the dissimilarities involved. If that difference is smaller than a fixed margin, the cluster is regarded as superfluous and its children will be added as separate entities to the parent of that cluster.

Finding these superfluous clusters needs to be done in a bottom up fashion, because this algorithm is more likely to remove coarse clustering steps high up in the hierarchy than fine grained clustering steps low in the hierarchy. Consider for example a group of three clusters, A, B and C, that ideally should be siblings of each other. The normal agglomerative algorithm will put two of these clusters, say A and B, in a new cluster, D, which will be grouped with C in another new cluster, E. If our post processing would be applied in a top down fashion, it might be decided that E is superfluous, joining D and C with the siblings of E. Now, whether or not D will be regarded as superfluous depends not anymore solely on the dissimilarity between A, B and C, but also between that of those clusters and the siblings of E.

For the implementation of this algorithm we modified the Hierarchical Agglomer-
ative Clustering library\(^7\) (HAC library) used in the tool Trevis [1] to include the post processing step described above. The complexity of this algorithm is \(O(n^3)\), as there are at most \(n\) clustering steps to be removed and the verification of each of these steps takes \(O(n^2)\), because all their direct and indirect children need to be compared with each other and with their siblings. Since the complexity of the HAC algorithm is also \(O(n^3)\), our algorithm only adds a constant factor. In practice this means that our algorithm will be run approximately twice as slow as the HAC algorithm, but on average it will be in the same order of magnitude.

### 4.2.3 Dissimilarity measure

Besides the algorithm itself, we also need to chose an appropriate dissimilarity measure. We came up with four different heuristics, from which only two were implemented due to time constraints. These heuristics are:

- Graphical dissimilarity
- HTML dissimilarity
- URL dissimilarity
- Menu hierarchy dissimilarity

**Graphical dissimilarity**

This measure was conceived by reasoning about the meaning of a clustering. In a web application, clusters should consist of states that have something to do with each other. Perhaps they present different instances of the same concept (like products in a webshop), or they may be steps in a specific work flow. In a well designed web application, such pages will look similar to each other in order to make clear to the user that the pages belong to each other. Humans naturally will pick up this visual hint and this is precisely what we want to mimic with the graphical dissimilarity: detect visual similarity and calculate the dissimilarity as the inverse of that similarity. Unfortunately, due to our unfamiliarity with the subject of image processing, implementing this metric would require too much time for the scope of this research, so we dropped this solution.

**HTML dissimilarity**

Because the visual appearance of a web page is entirely determined by the DOM of that page, visually similar pages may also have a similar DOM. This is especially true for web applications where the DOM is generated by a high quality framework, which will generate each type of component in a consistent way, and similar web pages are likely to be created using the same template. In order to measure the dissimilarity of two DOMs, we calculated the Levenshtein distance [33] between them. However, this proved to be far to inefficient, as the Levenshtein distance is at best a quadratic algorithm and the input consists of two entire DOM strings, which quickly reaches tens of thousands characters. With such inputs, the time required to calculate all pairwise dissimilarities takes an infeasible amount of time, even for small web applications. Thus, this solution was dropped as well.

\(^7\)http://sape.inf.usi.ch/hac
4. IMPLEMENTATION

URL dissimilarity

Based on the trend that URLs of modern web applications tend to be human readable and representing the navigational hierarchy of that application, the third dissimilarity measure we devised is based on the distance between the URLs of two states in the hierarchy extracted from those URLs. For example, the URLs www.example.com/company/team/andrew and www.example.com/company/team/david are two siblings of each other in the navigational hierarchy of that web site and should have a low dissimilarity (i.e., they are very similar to each other). On the other hand, the URLs www.example.com/company/team/andrew and www.example.com/projects/42 are relatively far apart from each other and should have a high dissimilarity. Next to the path, the other parts of the URL are also taken into account when calculating the dissimilarity, namely the scheme, the supplied user info, the port, the query string and the fragment. However, the path remains the most important factor, contributing for 50% to the end result. As this metric gave the best results, this was used for the evaluation of our tool. A drawback of this method, however, is that it only works for web applications that actually reflect the navigational hierarchy in the URL of each state.

Menu hierarchy dissimilarity

Similar to the use of the navigational hierarchy in the previously discussed URL dissimilarity, but more generally applicable, is to extract the navigational hierarchy from the menus found on the web site. Typically, a web site will have a menu allowing users to quickly access different parts of the web site. If this menu could be dynamically detected and interpreted, the resulting hierarchy could be used in the same way as the path was used in the URL dissimilarity. The advantage of this method over the URL dissimilarity is that it does not rely on the URL design and nearly all web applications have a menu somewhere. That immediately also constitutes the largest drawback: menus are implemented in a variety of flavors and a great deal of effort is needed to reliably detect the menu structure. Due to time constraints, we therefore skipped this metric, although we are very interested in how it would perform.

4.3 Test results

The implementation of the integration of test results is largely based upon the Automated DOM Invariant (adi) plugin developed for Crawljax in 2010 [44]. This plugin provides the functionality to generate regression tests, in the form of a set of invariants for each state, and the ability to verify these invariants (Section 4.3.1). We adapted this plugin by hooking it to Graphicrawl, in which we incorporated an extensible and transparent way to add invariant results from multiple sources (Section 4.3.2). Each source that adds its own type of invariant can then attach a UI to Graphicrawl. The UI for showing the DOM invariant results (Section 4.3.3) can serve as a reference implementation for future contributors.
4.3. Test results

4.3.1 DOM invariants
The DOM invariants are generated by crawling the system when that system is assumed to be correct, the so called ‘gold standard’. Based on a heuristic, a selection of elements present in the gold standard is made that should be present at all future crawls of the system, thus these elements are supposed to be invariant. The way in which these elements are identified, is by creating an XPath expression for each of these elements, targeting the same type of element and the same set of attributes. However, this might not uniquely identify the targeted element. For example, when the targeted element is a single `<li>`-element without attributes, it is likely to be found multiple times in the tested DOM, only one of which is the invariant element. But if the surrounding invariant elements can be found, then the ordering of the invariants can be used to determine which of the found `<li>`-elements is the one at the right position, between the previous and the next invariant. Therefore, the invariants are listed in a hierarchical way, the same hierarchy as that of the targeted elements in the DOM of the gold standard.

4.3.2 Invariant results in Graphicrawl
Figure 4.3 shows how the results of tested invariants are recorded in Graphicrawl. The GraphicrawlPlugin accepts any InvariantResult from any source that generates test results, by implementing the InvariantListener interface. Users can add their own implementation of an InvariantListener if they wish to process InvariantResults in a different way. In this way, all future automated testing approaches can easily add their own test results to Graphicrawl (by generating InvariantResults), and all future visualization approaches can record the test results as they like (by listening as an InvariantListener). In the case of the DomInvariantTester, it produces DomInvariantResults by checking if all InvariantElements, generated by the DomInvariantFinder, are present in the state that is currently being tested.

As mentioned in Section 4.1.1, the GraphicrawlPlugin stores these results as annotations in the AnnotatedSFG during the crawling phase. After the crawling phase, the AnnotatedSFG directs a post processing step, during which the DomInvariantTester tries to link the successful invariants to the exact position of the corresponding DOM elements in the HTML of the respective states. The fact that checking an invariant
element may yield a list of possible matches makes it impossible to create this link in
the crawling phase. In such a case, one can only locate the correct DOM element using
results of invariants that are, at that point, not yet checked.

4.3.3 Visualization of DOM invariant results

After the invariant results have been recorded, they need to be shown to the user. We
do this with the DomResultPanel, which implements an ExtensionPanel, show casing
how the UI of Graphicrawl can be extended with custom components. The panel shows
a filesystem like tree of the invariant hierarchy in which all succeeded invariants are
marked green and all failed invariants are marked red (see Figure 3.5). To quickly find
all failed invariants (often there are dozens of results being shown), the parents and
ancestors of each failed invariant are marked with a small icon, regardless of whether
that ancestor invariant itself succeeded or not. Another feature that speeds up the
process of locating the failed invariants, is that the user can collapse all items in the
tree except the failed invariants (and of course its ancestors).

When the user clicks on one of the succeeded invariants, a view below the tree
with invariant results will show the HTML of the state in which the invariant was
tested. Within this HTML, the DOM element to which the invariant was linked in the
post processing step, is highlighted. This only works for succeeded DOM invariants,
as by definition of these invariants, if they fail, there is no DOM element to which
they can be linked. As a compromise, the elements between the previous and the next
succeeded invariant can be highlighted, showing the user where the invariant element
should have been found. Due to time constraints, this feature was never implemented,
but as a workaround, users can manually select the surrounding successful invariants
and fictionally highlight the HTML in their minds.

4.4 Custom components

Because Graphicrawl only implements a limited set of features, and others may desire
the addition of similar yet different functionality, care has been taken to make Graphi-
crawl an extensible tool. This was done by adding five extension points in the code,
allowing third parties (e.g., a future developer) to modify or extend the behavior of
Graphicrawl. Some of these points have already been described earlier in this thesis,
all remaining extension points will be discussed here. The customizable features in
Graphicrawl are the clustering algorithm (Section 4.4.1), the edge filtering (discussed
previously in Section 4.1.3), custom views (Section 4.4.2), custom invariant results
(discussed previously in Section 4.3.2) and custom annotations (discussed previously
in Section 4.1.1).

4.4.1 Clustering algorithm

As discussed in Section 4.2, the currently used algorithm is our multi agglomerative
clustering algorithm (MAC). When setting up Graphicrawl to analyze a web applica-
tion, it can be instructed to use a custom algorithm, or to use a different dissimilarity
measure or linkage method in our algorithm. Below can be read in which ways this
can be achieved.
4.4. Custom components

Recall from Figure 4.1 that the Clusterer is responsible for the clustering. In Figure 4.4 can be seen how this Clusterer is further divided into different modules and how it interacts with the HAC library. The Clusterer is an interface that, given a graph, must return a hierarchy of the nodes that represents the clustering. To use another algorithm than the MAC algorithm, one can create his own algorithm without any restrictions, let it implement the Clusterer and attach it to Graphicrawl before analyzing a web application.

To facilitate the application of another dissimilarity measure or linkage method in the MAC algorithm, Graphicrawl contains its own implementation of the Clusterer that runs the MAC algorithm with customizable settings: the StateClusterer. The StateClusterer must be extended for each new implementation of a dissimilarity measure. As stated in 4.2.3, the HTML dissimilarity and URL dissimilarity have already been implemented. To use a different linkage method than complete linkage, one of the StateClusterer implementations can be extended (or adapted) and instructed it to use the AgglomerationMethod of choice. The HAC library used to execute the first step of the MAC algorithm provides seven different methods that can be used out of the box, including single and complete linkage.

4.4.2 Custom Views

Future contributers are given the opportunity to add completely new functionality to Graphicrawl or extend any current feature by adding new views on the fly to the UI. These views will show as a new tab next to the DOM errors tab (which are in fact examples of such custom views). Once added to Graphicrawl, they have instant access to Graphicrawl’s core data such as the clustering hierarchy of the states, the currently selected state or cluster and the AnnotatedSFG, to enable the view to retrieve all the information it needs. All one has to do is implement the ExtensionPanel interface and add that extension panel to Graphicrawl.

---

Figure 4.4: The clustering architecture

---

8 Actually called StateVertexClusterer, but for simplicity here called StateClusterer
Chapter 5

Evaluation Design

Now that we have implemented the proposed concepts, we can verify if the concepts answer the research questions by putting the tool to the test. As a quick reminder, the two main research questions in this thesis are “How can state-flow graphs of a web application be visualized to improve comprehension of the web application?” (RQ1) and “How can comprehension tools of web applications be enriched to support the locating of errors?” (RQ2). In this chapter, we will explain how we designed the study to evaluate the effectiveness of our tool and the concepts it implements.

5.1 Goal

The study serves two goals. First of all, we would like to know if the concepts we proposed contribute to the comprehension of web applications and the locating of errors, as meant in the research questions of this thesis. Second, we would like to know how Graphicrawl, which implements these concepts, makes its contribution. The two are not necessarily the same, since the implementation of a concept in Graphicrawl may lack an essential feature, or otherwise may spark more interest then expected.

As stated in Section 1.2, we assess the effectiveness of the implementation by answering the two subquestions “Does the state-flow graph visualization of Graphicrawl support developers in understanding web applications?” (RQ3) and “Does the visualization of DOM invariant results of Graphicrawl support developers with locating errors?” (RQ4). We will base the answers to these questions on feedback by (potential) users of Graphicrawl. While getting this feedback, we will also inquire the users about the higher level concepts in order to answer RQ1 and RQ2.

5.2 Evaluation method

Choosing the evaluation method is an important step in the design of the evaluation. It determines the value that can be attributed to the results. For our evaluation, we considered several alternatives and in the end decided to go with a contextual interview. Below we discuss the options we considered.

Our first thought was a controlled experiment, clearly showing the impact that Graphicrawl has on program comprehension and error locating. However, there are two factors that made this an infeasible choice. First of all, RQ1 and RQ2 are open
questions on purpose; we are mainly interested in fresh views on the concepts that we address. A controlled experiment will not dig up such ideas, but only quantifies the concepts that we ourself proposed. The second objection is of a more practical nature. Conducting a controlled experiment would require a large group of participants in order to get statistically valid results. Since we do not have the time, nor the resources, to conduct such a large scale study, we decided not to go for this type of validation.

Instead, we thought of a more exploratory user study, getting the information we need through interviews. This fits better to our objective of digging up fresh views, but does not allow us to validate the implementation of the concepts in Graphicrawl. Hence, we chose a contextual interview [30], in which the interview is accompanied by a demo or tutorial of the tool, providing the context for the interview. In this way, specific questions about the tool can be asked while at the same time the participants are given the freedom to give unconstrained feedback. This combines the strengths of both worlds: validating the effectiveness of Graphicrawl with the context and acquiring feedback on the concepts through the interview.

The way in which we conducted the interview is mainly infused by our wish to generate new ideas on the topics we present. Therefore, we used a semi-structured interview, taking a set of questions and running with it. The interviews were conducted in small groups of people, because experience with previous experiments showed that discussion between the participants often leads to new views on ideas.

5.3 Test setup

We divided the evaluation into four parts: a short questionnaire that determines the level of expertise of the participant in various areas; a tutorial that both explains the concepts implemented in Graphicrawl and guides the user through the tool itself; a questionnaire with which the user rated Graphicrawl and the concepts; and an interview in which we encouraged the users to elaborate on the answers given in the questionnaire. Below we discuss how we created these parts, the exact text that was given to the participants can be found in Appendix A

5.3.1 Knowledge level

The first part is a questionnaire that asks participants about their acquaintance with the topics relevant for this study, to verify that the participants indeed fit the profile for which we selected them (described in Section 5.5).

1: I might have heard of it, but I don’t know how it works or what it is
2: I’ve used it one time or another, but didn’t really look into it
3: I can use it, but I need a help page or a book next to me
4: I’m skillful, or I use it on a frequent basis
5: I’m an expert, skillful people consult me for my expertise

We selected ten topics for which they had to rate themselves. To evaluate their acquaintance with web applications in general, they were asked about HTML, DOM and FireBug (or similar browser plugins). To find out how well they would understand the test results, they were asked about XPath, testing and regression testing. To determine
that running the tool in its current premature state would not be a problem, they were asked about Java and the Eclipse IDE. Finally, in order to estimate their perception of the state-flow graph visualization, they were asked about graphs and hierarchical clusterings.

### 5.3.2 Tutorial

The next part is the tutorial itself. The function of this part is to make participants acquainted with our tool, so they could rate its effectiveness. But, as a secondary goal, it also allowed us to explain the concepts implemented in Graphicrawl to the user. Each time the user had to use a feature of Graphicrawl, they were first told about the concept that the feature implements. The advantage of that scheme is that users will never be clueless as to why they perform certain actions, making them appreciate a feature within a certain context. Furthermore, it allowed us to give some background information on the topic in which the feature was implemented, e.g., quickly explaining what an invariant is before directing the user to look at the hierarchy of invariants.

The tutorial started with a description of Graphicrawl: what problems it addresses and how it tries to solve them. Next, some background information is given. The user is explained what a state-flow graph is, how Crawljax crawls a web site, and the fact that clustering is applied.

Since Graphicrawl is still in an early stage of development, in which running the tool is done through the IDE, the user is first asked to simply open the Eclipse IDE. We have prepared the workspace of the tool, so the project and all dependencies are installed and fully configured.

Next, the participant may create his first crawl. A Java class, `SmallWebsite.java`, was pre-fabricated that configures and loads Crawljax and then loads Graphicrawl. This abstracted all the configuration away from the user, exposing only four descriptive parameters to the user. All he had to do was clicking ‘run’ and the crawl would be created. The tested web site was a small artificial site consisting of 5 pages with 18 links between them. When the crawling was complete, the GUI was opened, presenting Graphicrawl to the user for the first time, after which he was asked to perform several actions, showing all navigational features of Graphicrawl.

The next step covers clustering and how that looks like in Graphicrawl. We prepared a crawl from an example web site built with the open source CMS Contao\(^1\). It shows an HTML5 web site of a fictional music academy and includes some JavaScript to render lightboxes and accordion elements. Since a complete crawl of this website would require 10 to 30 minutes, this step uses a pre-created crawl that is loaded from local storage. This was done by composing the Java class `LargeWebsite.java`, which is analogous to `SmallWebsite.java`, but loads a different configuration and directs that configuration to load the crawl result from a file. After the GUI was started, the user was again asked to perform several actions which covered all navigational features concerning clusters.

The next step concerns the integration of regression tests. Note that in the previous steps, the crawls did not perform or include tests and consequently, the graph was not yet enriched with test results. However, the example of the small website did generate

\(^1\)http://contao.org, the used build is included in the source of Graphicrawl.
5. Evaluation Design

a test suite which will be used in this step. To this end, the participant had to alter
one parameter in SmallWebsite.java which changed the configuration to test the
invariants that were generated in the first step. After the crawl was made, the enriched
visualization was shown and the user was directed to use the features that come with
the test results.

In the last step, the user had to load a crawl in LargeWebsite.java in which
slightly more than 6000 invariants had been tested, 3 of which had failed. The user was
then guided through the analysis of one of these errors, after which an open exercise
was given to find out what went wrong with one of the other two failed invariants,
using the features of Graphicrawl.

5.3.3 Questionnaire

The questionnaire was presented to the participants directly after the tutorial and was
divided into four parts. Its questions were not designed to gain quantitative feedback
from the participants, nor to answer the research questions on their own, but instead,
they served as a starting point for the interview held immediately afterwards. The combi-
nation of the answers to the questionnaire combined with the results of the interview
enables us to answer all research questions of this thesis.

The first part asked participants to rate the user experience of Graphicrawl on a 5-
point Likert scale through a series of eleven statements about the implemented features
in order to get their opinions on these features. These questions focus on RQ3 and RQ4
and are the following:

1. I found Graphicrawl easy to use
2. Navigating through the state graph was intuitive
3. The highlights helped me with navigating through the graph
4. The preview helped me with navigating through the graph
5. The states that were clustered together belonged together
6. The graph was laid out in an intuitive way
7. The invariant hierarchy helped me understand what was tested
8. Linking the invariants to the HTML helped me understand what was tested
9. I could easily find out what went wrong with failed invariants
10. Graphicrawl can help me with understanding a web application
11. Graphicrawl can help me with testing a web application

The second part inquired the participants more generally about the concepts im-
plemented in Graphicrawl, by asking them to rate eight statements on a 5-point Likert
scale. The statements were either implicitly or explicitly\(^2\) detached from Graphicrawl
to encourage participants to think about the concept and rate how an ‘ideal’ implement-
ation would be. We expected that this would lead to interesting insights concerning
RQ1 and RQ2. The questions asked are:

12. A graph of a web application helps me understand its structure

\(^2\)Implicit: questions that ask about a certain feature. Explicit: questions that ask about ‘a tool like
Graphicrawl’
13. A graph of a web application helps me understand which sections roughly exist in that application
14. Clustering makes navigating through the graph more difficult
15. A tool like Graphicrawl is likely to save me time during testing
16. A tool like Graphicrawl is likely to save me time when I have to do maintenance for a web application I haven’t worked on before
17. A tool like Graphicrawl stimulates me to make my test suite better
18. A tool like Graphicrawl makes me more confident that I understand the web application that I’m testing
19. I would like to use a tool like Graphicrawl in a future project

At the end of these statements, the participants were asked in which development phase they would use a tool like Graphicrawl, so we could search for uses we did not anticipate. They could choose between design, development, testing, deployment or maintenance.

In the third part, we asked the participants to make a top three of the features of Graphicrawl. For the remaining features, they could mark those that they felt could be left out entirely. The purpose of this part is to get a qualitative ordering on the usefulness (and by extension the effectiveness) of the features in Graphicrawl. As stated in Section 3.3, the list of features is as follows:

- Having a visualization of the state graph
- Being able to see previews and full screen image captures of the states in the state graph
- Being able to navigate through the state graph via both the graph and the screen shots
- Clustering of the state graph
- Automatic and manual positioning of the nodes in the state graph
- Having a hierarchical overview of the invariants of the regression tests
- Linking the invariants to the HTML that satisfied these invariants
- Isolating the failed invariants in the invariant hierarchy

Finally, we used a more objective way to evaluate how Graphicrawl can help with program comprehension. A set of questions was selected from the questions stated by Sillito in 2006 [50], that programmers may ask themselves when they comprehend a program in order to be able to maintain it. We could only select a small set of those questions, because Sillito mainly focusses on reading and understanding source code, while the current version of Graphicrawl only investigates the presentation of the user interface of a web application. Thus, in the end we only selected the following six questions, where the appended question number refers to the numbering used in Sillito’s work:

20. What in this structure distinguishes these cases? (Q24)
21. Under what circumstances is this method called or exception thrown? (Q32)
22. How does the system behavior vary over these types or cases? (Q34)
23. What will be (or has been) the direct impact of this change? (Q42)
24. What will be the total impact of this change? (Q43)
25. Will this completely solve the problem or provide the enhancement? (Q44)
5. Evaluation Design

5.3.4 Interview

The interview was conducted in groups of two participants with the experiment leader, directly after the questionnaire. The experiment leader would look at the answers to the questionnaire and spot outliers or otherwise interesting answers. Based on their answers, the participants were asked to elaborate on them, perhaps explaining what they liked about a feature or why they thought a concept was not relevant for the comprehension of a web application.

5.4 Pilot study

Before performing the actual study, we did two pilot tests to evaluate the quality of the tutorial and questionnaire and to time the study. It also allowed the interview leader to practise with asking the right questions to get the most out of the participants. Measuring the time of the study was only done for a practical reason: during the recruitment of participants, we wanted to tell them how long the study would take. The test setup of the pilot differed with the actual evaluation in that only one participant at a time performed the pilot, leading to a less interactive interview.

The first pilot showed that some concepts needed a little more explanation, it was unclear to the participant how he had to interpret the questions selected from Sillito’s set, and some bugs were found in Graphicrawl. As for the interview leader, he noticed some difficulties in triggering the participant to elaborate on his answers from the questionnaire. After an analysis of that interview, several possibilities for improvement were identified: being more assertive, always focusing on the objective of acquiring elaborations to answers, and summarizing the participants feedback to find spots where more feedback is desired.

We fixed the bugs, adapted the text in the tutorial and questionnaire and were better prepared for the interview in the second pilot, which was performed without problems. The interview took slightly longer, but the other elements had the same duration. This allowed us to truthfully inform potential participants for the evaluation how long the study would take.

5.5 Participant selection

In order to invite participants to join our evaluation study, we sketched a typical profile of potential users of Graphicrawl and searched for people that fit this description. Graphicrawl is designed for developers of web applications who try to understand the structure of the user interface of such an application, or need to interpret regression tests for the user interface of a web application. Potential candidates therefore need to be web developers that are skillful in at least HTML and DOM manipulation, and ideally often work on web applications they have not seen before. Furthermore it is desirable that they have field experience and experience with (regression) testing.

Since the research is conducted at a university, the first group of people that came to mind are students, but students do not necessarily possess the targeted set of skills. However, some of them actively develop those skills in addition to their studies, often
having a job on the side or freelancing as a web developer. Therefore, we felt that students with such experience could provide valuable feedback as well.

A second group that fits the profile are professional freelance web developers. They have the right skill, often work on new projects, their experience is likely to be superior to that of students and they know the ins and outs of the field. However, they have less time available and are therefore less likely to cooperate than students. Therefore we decided to perform the study with as many professionals as we could reach and invite selected students if not enough professionals responded.

5.6 Evaluation study

We conducted the experiments in two groups with two participants each, where the participants worked through the tutorial and questionnaire individually, after which we all sat down with together for the interview.

The first group consisted of two students and performed the experiment at the university. We stayed with them in order to answer any of their questions or to take action should any bug occur, but did not interfere with them. In case they would discuss the features during the tutorial, we recorded the whole session, but no discussion arose so these recordings were not used for the interpretation of the results.

The second group consisted of two professionals and we performed the experiment at the office of one of them. During the experiment, one bug was revealed\(^3\), but this could quickly be fixed. Due to circumstances, they did not perform the tutorial at the same time, resulting in a pause of about two hours between the questionnaire and the interview for the first of the two participants. While he showed signs of having forgotten about some details, he was still able to accurately and deliberately give his feedback, showing that the results were still genuine and can be trusted.

---

\(^3\) A right-click was not recognized as the context menu trigger on Mac OS X Mountain Lion
Chapter 6

Evaluation Results

In this chapter we present and analyze the results retrieved from the questionnaires and the interviews. First, the profile of the participants is examined in Section 6.1, to be better able to interpret their feedback. Second, the results of the questionnaire and interviews are given and interpreted in Section 6.2. Finally, we analyse the threats to the validity of this research in Section 6.3.

6.1 Participant profile

As mentioned in Section 5.5, the evaluation study was performed with two students and two professionals.

The two students we selected were acquaintances that had done several web development projects, reaching mature development statuses. One of them is a current developer of Crawljax, and his feedback turned out to be very valuable, because of his knowledge of the inner workings of Crawljax and consequently that of Graphicrawl.

The two professionals both run their own company for several years already, specializing in web applications and, more recently, mobile applications. They are concerned with both the back end and the front end, with one of them having a tendency to the front end and one to the back end. They were invited by sending out emails to a handful of befriended freelancers, who passed it on to other people in their network. We received two replies of people interested in the tool and invited them to join the evaluation. Their professional interest in the tool makes them potential users of Graphicrawl and therefore perfect candidates.

The skill level of these four participants is depicted in Figure 6.1. As can be expected by their experience described above, all participants were (highly) skilled with HTML and DOM, showing they indeed fit our profile. Their experience with graphs should make them comfortable with our visualization, but a hierarchical clustering is quite new to them. However, we did not ask to their experience with clustering in general (as opposed to hierarchical clustering), so this low skill level might be misleading as to how they perceive our clustering. The XPath, testing and regression testing expertise are relevant for how they rate the usefulness of the test results incorporated in Graphicrawl, and it can be seen that the skill level is quite diverse.
6. Evaluation Results

Figure 6.1: Participants’ knowledge levels in the relevant fields. The values on the y-axis correspond to: 1 = “I might have heard of it, but I don’t know how it works or what it is”, 2 = “I’ve used it one time or another, but didn’t really look into it”, 3 = “I can use it, but I need a help page or a book next to me”, 4 = “I’m skillful, or I use it on a frequent basis”, 5 = “I’m an expert, skillful people consult me for my expertise”.

6.2 Result analysis

In this section we will present and analyze the results from the questionnaire and interviews. We will start with an analysis of the usefulness of Graphicrawl and its features for the comprehension of web applications in Section 6.2.1. Then the results on locating errors with Graphicrawl are discussed in Section 6.2.2. Next we present the top features according to the participants in Section 6.2.3 and finally, in Section 6.2.4, we will take a final look at how Graphicrawl can be employed.

The analysis of the results is guided by the answers to the questions from the questionnaire, and since these answers are mostly statements with which the participants could (strongly) agree or not, the analysis uses the words ‘question’ and ‘statement’ interchangeably. The answers are depicted by bar charts that show how many participants answered --, –, 0, + and ++. To make them easily comparable with each other, they are all of the same size and scale. Due to space considerations, each chart is accompanied by a minimal text that serves as a key phrase to distinguish it only from the other statements in the same figure. Each such text is followed by a number which is the corresponding question number in the questionnaire; the full text of the question will be mentioned in the analysis and can be found in Appendix A. On the author’s discretion, the order in which the questions are discussed is different from the order in which they were asked; however, all questions will be dealt with.
6.2. Result analysis

6.2.1 Comprehension of Web Applications

Interactive navigation

Figure 6.2 shows the opinions of the participants regarding the navigation. Three out of four participants could navigate through the state-flow graph intuitively (a). The one participant that disagreed, explained during the interview that the GUI fails to promote the available navigational features and hence is not intuitive. As an example he mentioned that he only found out after a long while that he could navigate by clicking on the highlighted elements in the screenshots.

His opinion was reflected by the results in (b), where he showed indifference towards the helpfulness of the highlights. The other participants, however, were all moderately positive, commenting that the highlights had the intended effect of showing where one navigates to when following a state transition.

The previews of the states were less appreciated (c). One participant mentioned that they were not recognizable enough. This relates to the visual similarity between the states; they looked far too similar to each other at the scaled size in order to distinguish them from each other. Since most web applications will have a consistent look and feel throughout the application, they will consequently suffer similarly from this issue. This suggests that the previews can either be removed, or must be enriched by a visual clue that emphasizes the subtle (visual) differences between the states.

One of the two positive participants replied that he liked the previews not for the actual image, but for the fact that the mere popping up of the preview provided the visual feedback he needed to easily navigate through the graph, quickly identifying which elements could be used for navigation and which not.

State-flow graph visualization

In Figure 6.3, it can be seen that the reception of the state-flow graph is moderately positive. The layout was found to be intuitive (a), with one participant remaining indifferent. Upon questioning, his rating turned out to be related with the clustering (see below). The other participants responded well to the dynamic and adjustable spring layout, although one participant would have liked to turn the automatic layout off.

Only half of the participants could get an understanding of the different sections in the web application by looking at the state-flow graph (b). One of the participants
6. **Evaluation Results**

![Figure 6.3: Questions concerning the graph visualization of the state-flow graph](image)

(a) Intuitive layout (6)  
(b) Understand sections (13)  
(c) Understand structure (12)

![Figure 6.4: Questions concerning the clustering of the state-flow graph](image)

(a) Semantically correct (5)  
(b) Difficult navigation (14)

commented that the different parts of the web application are not easily recognized in the graph and that it should more reflect the way the user perceives the application. Giving the states more descriptive names, instead of a sequence number like ‘State23’, would be a good start to solve this problem. As an example, he said that there was no simple way to directly spot the index page, even though it is the most important page of the application. A possible solution would be a top-down hierarchical graph visualization, which is likely to match the user perception better, clearly distinguishing the index page at the top of the hierarchy. The rest of the hierarchy might also reflect the user perception better, because the depth in the hierarchy matches exactly with the number of links the user has to click in order to reach that state. However, as discussed in 3.1.1, such a hierarchical layout introduces the problem of cross-links.

Understanding the structure with the state-flow graph was considered slightly more difficult, again with half of the participants responding that they were not particularly helped by the visualization (c). Both participants that neither agreed nor disagreed commented that a mere visualization of the state-flow graph is not enough to comprehend a web application. While it definitely helps you with comprehending the structure of the front end, the back end remains a black box and it is precisely this part that contains most of the application logic. They would have liked to see a link between the state-flow graph and the back end code, with one participant going as far as saying that without this link, the state-flow graph has no use at all.

**Clustering**

Figure 6.4 shows two faces of the clustering. On the one hand, the participants were moderately negative on the quality of the clustering (a), but overall they agreed that it eases navigation (b).
6.2. Result analysis

In the interview, we asked them what the problem was with the clustering, considering that they found it useful but not correct. Three participants found the clustering to be unclear and one of them explained the problem in more detail. He wondered what the rationales of the current clustering were, that the clustering mechanism should be more transparent, which is in line with the response that the graph does not reflect the user’s perception enough. It suggests that the clustering method must be shown in some way. With the current agglomerative clustering method, this means that the similarity thresholds could be shown, but this is still merely a mathematical property, which does not convey the semantics that the participant would have liked to see.

Instead, such semantics could be added by proper cluster names, which was another shared remark by the participants. With a small adaption to the clustering implementation it is possible to let the clustering method not only determine the cluster hierarchy, but also generate semantically meaningful cluster names. In this way, domain specific knowledge in the (custom) clustering implementation is used to add the semantics, which is commonly known as concept assignment and has been done previously in a semi-automatic way by [7, 3, 17]. Alternatively, several participants suggested that the user could be allowed to change the name of a cluster so the semantics can at least be added manually.

The interview also revealed two interesting positive comments. First of all, one participant mentioned that the clustering helps you to remember and visualize in which part of the application you are currently working. While focussing on your current topic of interest, this allows you to forget about the overall structure, because you can instantly remember it when looking at the state-flow graph. This was an interesting unintended effect, as the clustering was only added to reduce the visual clutter in the visualization of the state-flow graph. Apparently, it also helps with the comprehension of a web application by dividing the application into small chunks which can be comprehended independently of each other, reducing the amount of information that needs to be processed simultaneously.

Secondly, the same participant suggested that it is not necessary to show the entire graph in order to comprehend the structure of the front end. Consider for example a web developer that shows the structure of a client’s website to that client by means of the state-flow graph. The entire hierarchy is not what interests that client and instead, only those sections of the web application that are considered important should be shown, e.g., the main features of the web application. The clustering should be such, that a set of high level clusters coincides with these important sections and together with the integration of descriptive naming, this should lead to a more understandable graph visualization. This also implies that the user base of Graphicrawl is not limited to web developers, but also to clients of web developers.

Overall

Figure 6.5 shows how participants thought about Graphicrawl’s ability to help them with comprehending a web application in order to perform maintenance. While the opinions were divided on whether or not Graphicrawl would help them with comprehension (b), participants nevertheless uniformly agreed that a tool like Graphicrawl will make them more confident that they understand the application under investigation (c) and most found Graphicrawl easy to use (a). Three out of four participants
thought that a tool like Graphicrawl will save them time while maintaining a web application (d). These results suggest that the concepts implemented in Graphicrawl are positively received, but the implementation lacks in persuasiveness. A list of points for improvement can be found in Sections 7.1 and 7.3.

Questions programmers ask during maintenance tasks

The questionnaire included a list of typical questions programmers ask themselves when maintaining software, taken from Sillito [50], who defined 44 such questions. The participants were asked to indicate to what extent they agree that Graphicrawl can help them with answering those questions. As most of Sillito’s questions pertain to comprehension of the source code of an application, only six out of the 44 questions were selected. Below these questions are listed, numbered with the question number of this evaluation and accompanied with Sillito’s question number for reference.

20. What in this structure distinguishes these cases? (Q24)
21. Under what circumstances is this method called or exception thrown? (Q32)
22. How does the system behavior vary over these types or cases? (Q34)
23. What will be (or has been) the direct impact of this change? (Q42)
24. What will be the total impact of this change? (Q43)
25. Will this completely solve the problem or provide the enhancement? (Q44)

From the results (see Figure 6.6), it can be seen that there was no clear consensus regarding the first two questions (20 and 21). Distinguishing cases (a) was made possible, according to two participants, by the clustering hierarchy, in which each distinguishable case is represented by a different cluster. Graphicrawl can then help you to see in which case an error occurs, or in which case a state exists. On the contrary, one participant interpreted the question differently, where a case is a different method
6.2. Result analysis

Figure 6.6: Questions programmers ask during maintenance tasks that could be answered by Graphicrawl. Likert scale shows to what degree participants agree that Graphicrawl could help answering that question, appended numbers reference the question numbers in this thesis and in Sillito’s paper respectively.

to solve a certain problem or implement a feature. He then wanted to see how the system behaves under these different implementations, but he did not think Graphicrawl would be able to facilitate that.

Determining under what circumstances a method is called was in general thought to be not possible with Graphicrawl (b), which makes sense since that is closely related to the source code which is not included in Graphicrawl.

Analyzing the system behavior for different types or cases was considered more applicable to the uses of Graphicrawl (c). This can be explained with the fact that Graphicrawl is designed to get an overview of the entire web application. The user can then analyze the output of the web application (the states) for each different type or case, provided that he knows how the different types or cases map to a state in the state-flow graph.

Concerning the last three questions, (d) to (f), there were two prevailing opinions that were expressed in the answers and the interviews. One participant tended to disagree and three participants generally agreed that Graphicrawl would be able to help solve those questions. The one participant that disagreed explained that the used framework, Crawljax, is inherently nondeterministic and that therefore the difference between two runs of Crawljax before and after a change can never be attributed with absolute certainty to that change. The other participants, unaware of this limitation of Crawljax, were under the impression that, as a ‘symptom tool’, Graphicrawl is well suited to help with analyzing the direct and total impact and whether or not a change solves a problem.
6. Evaluation Results

Figure 6.7: Questions concerning the helpfulness of the display of the invariant results in Graphicrawl

6.2.2 Error locating

Invariant hierarchy

Figure 6.7 shows how participants rated the usefulness of Graphicrawl for locating errors, using the reports generated by the test plugin. Overall these features were positively received, with the strongest preference for the link between the invariants and the HTML.

The participants found that the invariant hierarchy helped them with understanding what was tested (a), although one participant pointed out an inconsistency with respect to the form of the XPath statements that define the invariants. Since they all start with `//`, it suggests that the asserted element does not have to be found at that exact point in the hierarchy. However, the implementation of the invariant checking enforces the correct hierarchical ordering of the asserted elements. Hence, that participant did not agree that the invariant hierarchy helped him understand what was tested. The same participant thought that the entire invariant hierarchy was not needed for the purpose of error locating. He found that it is suffices to show the failed invariants, the rest is merely clutter. This was contradicted by the other participant in the interview, saying that it is precisely because of the hierarchy that you can easily locate the errors. Lastly, one participant would have liked to see a list of all generated invariants in Graphicrawl before they are tested, to be able to see what will be tested in subsequent runs.

Figure (b) shows a strong liking for the feature that links the succeeded invariants to the actual HTML. All participants found it helpful to understand the generated test suite and indicated that the highlighted HTML clearly showed what caused the invariants to succeed. Three participants did mention a point of improvement, namely that more information should be given on failed invariants. A distinction should be made between an invariant element that is completely missing and one that is malformed. In the latter case, the element should then be highlighted in the HTML, accentuating the attributes that caused the invariant to fail. However, currently, the test framework did not make this distinction between missing and malformed invariants, so they could not be highlighted in Graphicrawl. An improvement that could be made within the boundaries of the current limits is that an approximation of the location of the error can be made, for example highlighting the HTML where the invariant should have been found.

The ease with which these features were used is reflected by (c), in which all but
6.2. Result analysis

Figure 6.8: Questions concerning the features of Graphicrawl for supporting automated testing

![Bar chart showing help with testing (11), saves time (15), and stimulates testing (17)]

one participant indicated they could easily find out what was wrong with the failed invariants in the example web application. The one participant that neither agreed nor disagreed with that statement commented that it must be possible to analyze failures by selecting only that invariant, and not by investigating the surrounding succeeded invariants (by using the methods discussed in the previous paragraph).

Usefulness for testing

Figure 6.8 shows what participants thought of the usefulness of Graphicrawl for testing web applications. During the interview, one participant pointed out that he would like to be able to test JavaScript. With an automated test plugin readily available for Crawljax and the modular system from Graphicrawl, it requires little effort to add this functionality to Graphicrawl. One only has to adapt the test plugin to record all test results in Graphicrawl and a GUI element that displays these results. Another participant mentioned that the tests need to be more robust with respect to dynamic content. Instead of verifying that certain elements must be present, it should detect lists and other repetitive structures that can consist of zero or more items. This, however, is an improvement that should be made in the test framework and cannot be fixed by Graphicrawl. On the positive side, the participant that currently works for Crawljax could tell that these features are planned for the near future.

Figure (a) shows that half of the participants think Graphicrawl can help them with testing a web application. We observed that these were the participants that indicated to have only little or no experience with testing. One of them commented that the visual indication of the location and existence of errors is very helpful, and he recalled a particular use case that he recently encountered where he thought this would be a significant improvement. It concerned the monitoring of a content management system that is deployed to many different customers, but the content is still managed by himself to make sure it is entered correctly and does not deform the front end. Graphicrawl could be a valuable addition, where tests would be run on a daily basis in order to detect malformed content and the visualization could be a dashboard that he can use to quickly identify the sites that cause problems. In that scenario, the responsibility of adding new content can be transferred to the customers themselves, relieving the administrator.

Participants were less inclined to think that a tool like Graphicrawl will save them time during testing (b). Similarly to (a), it were the participants with little experience
6. Evaluation Results

<table>
<thead>
<tr>
<th>Feature</th>
<th>P1</th>
<th>P2</th>
<th>P3</th>
<th>P4</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1: Visualization of state-flow graph</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>F2: Previews and screen shots of states</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F3: Navigate via the graph and the screen shots</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4: Clustering of the state-flow graph</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>F5: Semi-automatic layout of state-flow graph</td>
<td></td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>F6: Invariant hierarchy</td>
<td>3</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F7: Linking invariants to HTML</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8: Isolating failed invariants</td>
<td>1</td>
<td>3</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6.9: Participants’ top 3 features, each column represents one participant. Gold cells represent their top choice, silver cells their second choice, bronze cells their third choice and red cells mark features they found useless.

with testing that agreed to the statement. The participant that disagreed said that he did not know how much effort it would require in order to adopt Graphicrawl in his testing habits and consequently did not agree. For example, he would like to automate the test process and add it to his build cycle, but did not know if this was possible. In fact, this is possible and Graphicrawl can even be configured to only show the GUI if errors were detected. Of the participants that agreed, one said that front end testing of web applications is still rarely automated and thought that Graphicrawl could make a change to that. The other participant that agreed with the statement said that cross-browser testing costs the most time and if Graphicrawl were configured to run the tests in multiple browsers, it would be a real time saver. When the work of Lenselink [31] for Crawljax is incorporated into Graphicrawl, this is exactly what the participant wanted.

In Figure (c) it can be seen that participants thought that a tool like Graphicrawl would stimulate them to make their test suite better. The participants did indicate that it depends on whether or not such a tool facilitates an easy way to create those tests. The participant that did not agree, did so because of this uncertainty. He commented that it is not transparent how Graphicrawl creates the regression tests and hence he cannot know if and how he can create his own test suite, let alone that it stimulates him to do so.

### 6.2.3 Top Features

The participants were asked to rate a selection of the features from Graphicrawl and to indicate if they found any of these features unuseful. The features they could chose from were:
6.2. Result analysis

Figure 6.10: Using Graphicrawl. The abbreviations in (b) stand for Design (Des), Development (Dev), Testing (Tst), Deployment (Dep) and Maintenance (Mnt)

F1: Having a visualization of the state-flow graph
F2: Being able to see previews and full screen image captures of the states in the state-flow graph
F3: Being able to navigate through the state-flow graph via both the graph and the screen shots
F4: Clustering of the state-flow graph
F5: Automatic and manual positioning of the nodes in the state-flow graph
F6: Having a hierarchical overview of the invariants of the regression tests
F7: Linking the invariants to the HTML that satisfied these invariants
F8: Isolating the failed invariants in the invariant hierarchy

Figure 6.9 shows how the participants rated the features. There are two things that immediately catch the eye: F8 is clearly the most appreciated feature and there are two reasons for concern because of features that were marked unuseful. During the interview, an explanation was given for the unusefulness of these features.

Concerning F1, it turned out that the participant did not find that feature particularly unuseful, but rather unclear. The reasons for this were the same as the criticism given in 6.2.1 on the state-flow graph and the clustering.

Concerning F6, the participant gave the reasons stated in 6.2.2, namely that the entire invariant hierarchy was not needed for the purpose of error locating, although the presence of the feature did not bother him. As said previously, his opinion was contradicted by the other participant in the interview, saying that the hierarchy is needed to locate the errors.

Regarding the top features, there are two features that stand out above the others. With three first choices and one third choice, F8 is clearly the most appreciated feature and in second place comes F7, being someones first, second and third choice. A likely explanation for the popularity of F8 is that it eliminates the need for them to search through the hierarchy for the errors, which is consistent with the criticism of the participant that found F6 unuseful. Similarly, F7 eliminates the need to search through the HTML to find the tags that satisfy the selected invariant, which is necessary to find the location of succeeded invariants around a failed invariant to estimate the location where the failed invariant should have been found.
6.2.4 Applicability

In order to broaden the applicability of Graphicrawl beyond the applications anticipated in this thesis, the participants were also asked to indicate if and how they would employ Graphicrawl. This led to interesting insights, of which two have already been mentioned: using Graphicrawl to monitor a diverse range of systems and using Graphicrawl as a tool to communicate a high level view of a web application to customers.

In Figure 6.10, participants were asked if they would like to use a tool like Graphicrawl in a future project (a). They responded positively to this, showing that they recognize the usefulness of its features. They were then asked to indicate in which phase of such a project they would employ Graphicrawl (b). None of the participants would use it during the design of an application, which makes sense since there is nothing to comprehend or test at that point. Half of the participants would employ it during development and testing, which is an unexpected result considering the popularity of the testing features. Instead, participants were more inclined to use Graphicrawl during deployment and maintenance. Upon asking, the deployment phase was chosen to verify that the product passes all regression tests before deploying it, which comes down to a testing procedure.

6.3 Threats to validity

In this section we will discuss which factors were involved that may have led to misleading results, i.e., which may have threatened the validity of this research. The internal validity (Section 6.3.1), the external validity (Section 6.3.1) and the construct validity (Section 6.3.1) will be distinguished.

6.3.1 Internal validity

The internal validity refers to the confidence that our results were not caused by other factors than given by our interpretation. We identified several threats to this validity in the experimental settings, listed below.

The participants might have favored positive answers over negative ones to make the results look good, meaning the answers were caused by their kindness rather than by their valuation of Graphicrawl. This was mitigated by stressing the importance of honest answers over positive answers.

Similarly, the phrasing of the questions, both during the questionnaire and the interview, might have influenced the answers of the participants. For example, the interviewer may have worded questions in such a way that a participant is pushed to give a certain desirable answer. We tried to mitigate this by posting questions as neutral as possible and by giving plausible examples for both sides of a discussion to illustrate there is no single correct answer.

A threat to the validity of negative responses is that they may have been caused by the limitations of Graphicrawl, rather than the limitations of the concepts that those questions address. This concern was lessened by the addition of the interviews, in which the participants were asked to elaborate their opinion and by asking questions about both Graphicrawl itself and a tool like Graphicrawl.
6.3. Threats to validity

6.3.2 External validity

The external validity refers to the confidence that our findings apply to other settings as well, i.e., that the results are reproducible.

The evaluation was conducted with four persons, which is not enough to be meaningful due to individual differences. To reduce this risk, the people were selected from four different perspectives, leading to as much spread as possible in the population of this study. Another risk is that the participant selection was based on our network, which may have influenced them by means described in the threats to internal validity. We have made no active attempt to mitigate this, since half of the participants were in fact unknown to us, eliminating this concern at least for those participants.

The tutorial that had to be carried out by the participants only analyzes one web application, which possesses only a few characteristics of a web application (compared to regular web sites). Thus, the results may very well be different if another web application is analyzed in the tutorial.

6.3.3 Construct validity

The construct validity refers to the confidence that our methods of measurement are indeed measuring what we wanted to measure. In other words, are our methods of measurement correct? Several possible threats to this type of validity have been identified and are listed below.

To measure the adequacy of the concepts introduced in Chapter 4, the tool Graphicrawl was implemented and the participants were asked to give their opinion based on their experience with Graphicrawl. However, Graphicrawl only shows one instantiation of the concepts, biasing the measurement to only this instantiation, commonly known as mono-operation bias. In particular, there is only one implementation of the following features: the graph layout; the clustering; the edge filtering and the type of test reports that are visualized.

A second threat to construct validity was identified in the fact that Graphicrawl and the concepts it implements are meant to help web developers with comprehending web applications, but in fact helps web developers with comprehending state-flow graphs of web applications, which is only a small subset of the comprehending of web applications. This is known as construct confounding.
Chapter 7

Conclusions and Future Work

Based upon the evaluation we conducted (Chapters 5 and 6), the research questions stated in the introduction (Chapter 1) can now be answered (Section 7.1). This is followed by a summary of the contributions of this work (Section 7.2) and the identification of possibilities for future work (Section 7.3).

7.1 Conclusions

Within the field of program comprehension, we aimed to improve the comprehension of web applications by visualizing their state-flow graph in a dynamic visualization enriched with annotations. One of these annotations is the visualization of test results in this state-flow graph.

A tool was created (Chapters 3 and 4) to evaluate (Chapters 5 and 6) the effectiveness of several proposals (Section 3.1) that were candidates to answering RQ1 and RQ2 (for the research questions, see below). The evaluation of the tool was guided by RQ3 and RQ4, whose answers, together with all other feedback acquired during the evaluation, enabled us to answer RQ1 and RQ2. Thus, in the following text, we will first answer RQ3 and RQ4 and then RQ1 and RQ2.

RQ3 Does the state-flow graph visualization of Graphicrawl support developers in understanding web applications?

In short, the visualization of the state-flow graph in Graphicrawl was received positively, but not enthusiastically, and the opinions are divided on the helpfulness for comprehending web applications, leaving the answer somewhere in the middle, but certainly not a conclusive yes. Several pointers were identified to improve the visualization, which are given in Section 7.3. Future research is needed to see if Graphicrawl will be able to support the understanding of web applications when those improvements are implemented.

In more detail, the answer to RQ3 can be divided into several different insights:

- The graph layout adds to the comprehension of the front end of a web application, but gives little to no insight into its back end.
7. Conclusions and Future Work

- The clustering method resulted in clusters that were hard to relate to the web application, but would have been useful for the comprehension had a proper clustering method been used.
- Highlights in the graph had added value to the interactive navigation, but the previews of states did not add value to the navigation.

Besides these insights, we have also gathered evidence that Graphicrawl could be used to answer three typical questions that were proposed by Sillito [50]: 1) What will be (or has been) the direct impact of this change? 2) What will be the total impact of this change? and 3) Will this completely solve the problem or provide the enhancement?

**RQ4** Does the visualization of DOM invariant results of Graphicrawl support developers with locating errors?

In short, the integration of DOM invariant results in the state-flow graph was positively received and the participants indicated it does indeed help them with quickly locating violations of DOM invariants. However there is still room for improvement, as can be seen further down in Section 7.3.

In more detail, the answer to RQ4 can be divided into several different insights:

- Annotating the state-flow graph visualization with the invariant results was considered helpful for locating errors.
- Showing a hierarchical overview of the invariants gives a clear overview of the generated test suite.
- Isolating failed invariants and linking the invariant results to the HTML are highly appreciated features by developers.

These answers are interesting for an evaluation of Graphicrawl, but for broader conclusions we have also asked the participants to take a step back and discussed the concepts with them. Based on the experiences with Graphicrawl and these discussions we can now answer RQ1 and RQ2.

**RQ1** How can state-flow graphs of a web application be visualized to improve comprehension of the web application?

We have identified several guidelines that serve as an answer to RQ1. These should be taken into account when visualizing state-flow graphs for the purpose of comprehending web applications and are summarized below.

- Visualizing a state-flow graph with a force-directed layout is an appropriate method, but important nodes such as the index must be emphasized.
- The visualization should be linked to other artifacts, such as the source code, as the graph by itself only gives insight into the front end and not the back end of the web application.
- Clustering the states in a state-flow graph contributes to a clear visualization only if the resulting clustering closely resembles a user’s perception of the structure of the web application.
7.2. Contributions

- In order to comprehend the structure of a web application at a high level, it is only necessary to show nodes that are close to the initial state start page.

RQ2 How can comprehension tools of web applications be enriched to support the locating of errors?

Similar to RQ1, we have identified several ways in which a tool for the comprehension of web applications can be enriched to support the locating of errors, serving as an answer to RQ2. They are listed below.

- In order to quickly determine in which state an error occurred, the test results can be integrated into a visualization of the state-flow graph. However, if the responsible code needs to be located, integration into the state-flow graph will not have added value.
- If test results can be linked with the HTML of the web application, making this link will highly contribute to the locating of errors. If this link exists but cannot be made, an educated guess should be made.
- If the comprehension tool has an overview of the web application, this can be successfully used to show the density of errors, i.e., which parts of the application contain the most errors.

7.2 Contributions

This work contributes to the research towards comprehension of web applications in the following ways:

- The proposal of several concepts by which the visualization of state-flow graphs of web applications can be enhanced to improve their comprehension. These concepts addressed the graph layout, user interaction, navigation, clustering and abstraction.
- The proposal of several concepts by which comprehension tools for web applications can be enriched to support the locating of errors. These concepts addressed error locating and test suite comprehension.
- An implementation of the proposed concepts in a proof of concept called Graphicrawl, aimed at evaluating those concepts, which can be downloaded at https://code.google.com/p/in5000jfreisen. The tool is made extensible so that future contributors can add their own concepts or improve our proposed concepts.
- An extension to the agglomerative hierarchical clustering algorithm that differs from the existing algorithm in that it allows more than two clusters or nodes to be agglomerated per step.
- An evaluation study by means of a contextual interview in which the context was provided by Graphicrawl.
7.3 Future work

As was already mentioned before, we are convinced that there exist many more opportunities to be researched, and the evaluation study also provided several suggestions for improvement. Here, we will list these features, followed with a list of interesting topics that were not evaluated in this thesis, but do deserve academic attention.

7.3.1 Tool extensions

The following features could be added to or changed in Graphicrawl in the future:

- Different layout algorithms to match the user perception of the structure of the analyzed application. When proposing to use a force-directed layout, we already mentioned that a hierarchical layout could be suitable as well and this was also mentioned during the interviews. Furthermore, there may be more suitable force-directed layouts available, such as the one presented by Gansner and North [23].

- Different clustering algorithms. Similar to the layout algorithm, another clustering algorithm could yield better results. A plethora of algorithms exist, see for example the survey of Schaeffer [46].

- Different (dis)similarity measures and other parameters. We already proposed two different measures that we did not implement, but since these measures are all heuristics, plenty of different methods can be devised.

- Different edge filtering techniques, in particular a domain independent one.

- More recognizable visualizations of states in the nodes of the graph, especially for the clusters, to quickly identify which clusters or node corresponds with which section or state.

- Apply concept assignment by either manually or automatically assigning meaningful names to the clusters and states.

- The option to turn the automatic layout off and to remember the layout between successive runs of the tool, as a backup should the layout algorithm remain unsatisfying.

- Adapt the previews when hovering over states to emphasize the differences between them, because most pages in a web application tend to resemble each other.

- Link failed DOM invariants to the HTML as well by making an educated guess. A distinction should be made between missing DOM elements and malformed DOM elements.

- Integrate more types of test reports into Graphicrawl, such as JavaScript tests.

- Show the generated test suite before the test suite without executing it.
7.3.2 Evaluation topics

To get a better idea of the usefulness of Graphicrawl’s features, the following topics should be evaluated as well:

- Evaluate the effectiveness of edge filtering.
- Perform a pre-posttest in which the participants are required to work on comprehension tasks to quantify and better qualify the added value of Graphicrawl.
- Perform the evaluation with more participants to get statistically significant results.
- Apply Graphicrawl in a real world program comprehension use case.
Bibliography


Appendix A

Evaluation Documents

For the evaluation the participants received a document that consisted of three parts, a questionnaire to assess the participants knowledge level, a tutorial about Graphicrawl and a questionnaire about Graphicrawl. In the following sections, the text is displayed exactly as given to the participants, except that the questionnumbers were not present in their version, but are essential for referencing within this thesis.
A.1 Knowledge Level

First of all, thank you very much for helping me with the evaluation study for my thesis. During this evaluation study, you will be asked to do a small tutorial for a tool called Graphicrawl, an explanation of which will follow in the next step. Before we continue to that part, I would like to know your acquaintance with the topics relevant to this study. In the list of topics below, please indicate for each of them how familiar you feel with that topic on the following scale:

1 = I might have heard of it, but I don’t know how it works or what it is  
2 = I’ve used it one time or another, but didn’t really look into it  
3 = I can use it, but I need a help page or a book next to me  
4 = I’m skillful, or I use it on a frequent basis  
5 = I’m an expert, skillful people consult me for my expertise

<table>
<thead>
<tr>
<th>Topic</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XPath</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FireBug or similar browser plugins</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eclipse IDE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Graphs (nodes and edges, not charts or pictures)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical clustering</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Regression testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A.2 Graphicrawl Tutorial

In this tutorial you will become familiar with the features of Graphicrawl. Graphicrawl is a tool that was built to improve the comprehension of the structure of a web application, by interactively visualizing the navigational graph of the web app. Furthermore, it helps developers with regression testing by color coding the graph, based on the errors that were detected by the framework upon which Graphicrawl was built, Crawljax, and it will give an overview of the regression tests that were conducted to find those errors.

How it works

Graphicrawl uses the notion of states to represent the nodes of the navigational graph. A state consists of the top level DOM object of a page. Thus, every page on the website is another state, but also every change within a page leads to a new state. Think for example of a widget that can be hidden when clicked upon: even though it doesn’t navigate to a new page, it will still lead to a new state because the DOM of that page has changed (most likely the visibility attribute of a div element has changed). Two nodes in the navigational graph are linked by a directed edge if you can click an element on the first page such that you will be taken to the second page. This also holds for changes within a page.

When Graphicrawl is started, it will open a website and try to find all possible links on that site. All links that point to another page on the same website, or change the DOM in any way, are clicked until all pages (read: states) have been seen. The resulting graph is shown in a GUI in which you can analyze the graph. To reduce the visual clutter that may occur in dense graphs, similar states are grouped together.

Preparation

Graphicrawl is still in a stage of development and doesn’t have a configurable GUI yet. In order to change a setting, you will have to change a line of code and restart the app. Therefore, Graphicrawl will be started from an IDE. This means that you will have to make small edits to the code during this tutorial. I have prepared a workspace in Eclipse on your workstation that is completely configured to let you work with Graphicrawl without further ado. Thus, you can start by starting Eclipse. We will do all our work in the default package of the project “Graphicrawl”.

Creating your first crawl

A crawl is the binary output that Graphicrawl produces while visiting all the pages of the website. I have prepared a class that makes a crawl of a small example website, but before we start, let’s take a quick look at the website so we know what we’re working with. Open your favorite browser, browse to http://localhost/testsuite/crafted-1 and click around for a bit. Then, switch to Eclipse and open the file SmallWebsite.java in the default package and run it as a Java Application. It is configured to crawl the example website, generate invariants for the regression tests and show Graphicrawl afterwards. During the crawl you will quickly see webpages flashing in and out of Firefox. That is Crawljax giving commands to the browser to click links, make screenshots and extract the DOM. During this phase, try to minimize use of the keyboard, or at least don’t press any meta keys (control, alt, shift, etc.), because they will be consumed by Firefox and may corrupt the crawl.
A. Evaluation Documents

Using the Graphicrawl GUI

After the crawl has been made, the GUI of Graphicrawl will pop up, showing you the navigational graph on the left side and extra information on the selected state on the right side. In the small example we use here, nodes will not have been grouped together into clusters. Edges that lead to a node that has many incoming edges will be displayed in a lighter shade of gray, based on the assumption that such edges are less important than those that go to states with few incoming edges. Try the following actions and continue to the next part:

- Hover over a node in the graph. A preview of that page will be shown.
- Select a node in the graph by left-clicking on it. A screenshot of that page will be shown.
- Hover over a node in the graph while another node has been selected. If you can navigate to that node by clicking a link in the currently selected node, that link will be highlighted in the screenshot. Note in that case that the preview is positioned next to the link.
- Reposition a node in the graph by dragging it. The other nodes will try to adjust their own position so they fit nicely around the node you just repositioned.
- Hover over a link in the screenshot of the selected node on the right, then click it to go to that node. See how the highlighting of nodes and edges in the graph help you with keeping track of your navigation.

Clustering of nodes

Now that we’ve seen the navigational features of Graphicrawl, we can try a somewhat larger website, with more states. A drawback of displaying a graph with a large number of states is that the graph may become cluttered. This has been solved by clustering: similar states and/or clusters are grouped together into clusters, which are being displayed as a special node in the graph. Clusters can also be clustered, leading to a hierarchy of clusters.

Because crawling a website with many states can take quite some time, I have prepared a crawl for you that can be loaded from disk. But, just as with the small website, let’s start with a quick look at this website before we continue. Browse to http://localhost/testsuite/dolphin and click around for a bit. Open the file LargeWebsite.java in the default package and run it as a Java Application.

In this example, an edge filter is applied, which filters out edges that do not contribute to the comprehension of the structure, such as the edges that represent the links in the menu structure: every page contains this set of edges, but it suffices to show them only for the index page. This edge filter is created manually, so the decision which edges should be filtered out lies with the programmer. Note that, in the code from LargeWebsite.java, the parameter that used to say “FIND_INVARIANTS” now says “NONE”, meaning that the crawl will not be executed, but loaded from disk. Now, try the following actions:

- Identify the appearance of a cluster: round and slightly larger than normal nodes.
- Hover over a cluster in the graph. Previews of several states in that cluster will be shown on top of each other.
- Select a cluster by left-clicking on it. Thumbnails of all states in that cluster will be shown.
- Hover over a thumbnail on the right, then click on it to navigate to that state.
- Double-click a cluster to expand it, or right-click that cluster and select ‘Expand’.
- Right-click a node or a cluster and select ‘Expand all’. You will see the graph as it is without clustering.
- Try to arrange the leaf nodes in a sensible way, but don’t put too much effort in it.
- Right-click a leaf node, go to ‘Collapse’ and choose a set of nodes to collapse.

This concludes the navigational part of Graphicrawl. The following part will show you how the results of regression tests are displayed by the tool.

**Regression testing a website**

The framework Crawljax is capable of finding and running regression tests on websites. This means that it will first find invariants that should always hold, after which these invariants will be tested in subsequent crawls of that website. Invariants in this case are a list of elements that must be present in the DOM of a page, and they must be present in the given order.

In the first exercise, SmallWebsite.java, the example configuration was created with one of the arguments set to Crafted1.FIND_INVARIANTS. This instructed the crawler to generate the invariants during that crawl. Now, close Graphicrawl if you haven’t already done so, and go back to the editor of SmallWebsite.java. Change the argument from Crafted1.FIND_INVARIANTS to Crafted1.CHECK_INVARIANTS, to instruct the crawler that the generated invariants must be tested. Note that the path to the directory in which Graphicrawl places its output must be the same as when the invariants were generated. In this example, this has already been set correctly for you. Then, run the example again. When Graphicrawl is shown, the first thing you will notice is the green overlay on the nodes. This means that all tests in all nodes have succeeded. Try the following actions:

- Select a node and go to the DOM errors tab on the right side.
- Take a look at the list of invariants and note that they all passed.
- Select an invariant. The part of the DOM that satisfied this invariant will be highlighted in the HTML below.

In the next part we will see how we can identify errors when one or more invariants failed.

**Analyze errors in regression tests**

For this last part, we will use the prepared example of the large website again. Close Graphicrawl and go back to LargeWebsite.java. The parameter we will change is the directory from which the crawl will be loaded. We will load another crawl that contains the executed regression tests. Change the directory string from “Large website (without tests)” to “Large website (with tests)”. Now, run the application and you will immediately see note that some clusters are highlighted in red. This means that at least one invariant failed in that cluster. Now complete the following actions (see next page):

- Select the clusters called ‘state17 and 36 more’ and go to the DOM errors tab on the right. You will see heaps of invariants, most of which passed.
- Right-click any line and choose ‘Expand only failures’. Only those branches in the invariant hierarchy that contain a failure are expanded.
A. EVALUATION DOCUMENTS

- Select a failed invariant. Nothing will be highlighted in the HTML below, since the invariant was not found in the HTML.
- Select the invariants below and above the failed invariants and see which element is missing, or which element should have been found and why it wasn’t found.

A line displaying an invariant in the DOM invariant hierarchy ends with an XPATH expression that tries to finds the correct element in the DOM. If that element is found, the invariant is satisfied, otherwise, it fails.

- Look at a failed invariant’s XPATH expression and compare it with the HTML where you would expect the failed invariant to have been found. See if you can determine why the XPATH expression did not yield the element it was looking for.
- Select the cluster called ‘state26 and 1 more’ and try to identify the error yourself.
A.3 Questionnaire

Thanks for completing the introduction to Graphicrawl. You will now be presented with a questionnaire to evaluate the usefulness of the tool, which will be followed by a short interview. Please keep in mind to answer the questions in this questionnaire as faithfully as possible, try not to favor positive answers over negative ones to sooth me.

User experience of Graphicrawl

The following statements concern the user friendliness and effectiveness of Graphicrawl. For each of the statements, please indicate to what extent you agree with it, ranging from 1 (completely disagree) to 5 (completely agree).

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>I found Graphicrawl easy to use</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Navigating through the state graph was intuitive</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The highlights helped me with navigating through the graph</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>The previews helped me with navigating through the graph</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>The states that were clustered together belonged together</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>The graph was laid out in an intuitive way</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>The invariant hierarchy helped me understand what was tested</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Linking the invariants to the HTML helped me understand what was tested</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>I could easily find out what went wrong with failed invariants</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Graphicrawl can help me with understanding a web application</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Graphicrawl can help me with testing a web application</td>
<td>□ □ □ □ □</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A. EVALUATION DOCUMENTS

Features implemented in Graphicrawl
The following statements concern the features that were implemented in Graphicrawl, but apply to any hypothetical program that would implement the similar features. For each of the statements, please indicate to what extent you agree with it, ranging from 1 (completely disagree) to 5 (completely agree).

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>A graph of a web application helps me understand its structure</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>13</td>
<td>A graph of a web application helps me understand which sections roughly exist in that application</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>14</td>
<td>Clustering makes navigating through the graph more difficult</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>15</td>
<td>A tool like Graphicrawl is likely to save me time during testing</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>16</td>
<td>A tool like Graphicrawl is likely to save me time when I have to do maintenance for a web application I haven’t worked on before</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>17</td>
<td>A tool like Graphicrawl stimulates me to make my test suite better</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>18</td>
<td>A tool like Graphicrawl makes me more confident that I understand the web application that I’m investigating</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
<tr>
<td>19</td>
<td>I would like to use a tool like Graphicrawl in a future project</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

Furthermore, please indicate in which phase of a project you would use a tool like Graphicrawl (regardless of the answer to the previous statement). Multiple answers are possible.

☐ Design     ☐ Development     ☐ Testing     ☐ Deployment     ☐ Maintenance
**Top Features**

Below are a number of features of Graphicrawl. Please mark the features that you didn’t find useful. From the remaining features, please select your top 3 features. Put a “1” next to the best feature, a “2” next to the second best, and a “3” next to the third best feature. If you have less than 3 features remaining as useful, rank only those features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Having a visualization of the state graph</td>
<td></td>
</tr>
<tr>
<td>Being able to see previews and full screen image captures of the states in the state graph</td>
<td></td>
</tr>
<tr>
<td>Being able to navigate through the state graph via both the graph and the screen shots</td>
<td></td>
</tr>
<tr>
<td>Clustering of the state graph</td>
<td></td>
</tr>
<tr>
<td>Automatic and manual positioning of the nodes in the state graph</td>
<td></td>
</tr>
<tr>
<td>Having a hierarchical overview of the invariants of the regression tests</td>
<td></td>
</tr>
<tr>
<td>Linking the invariants to the HTML that satisfied these invariants</td>
<td></td>
</tr>
<tr>
<td>Isolating the failed invariants in the invariant hierarchy</td>
<td></td>
</tr>
</tbody>
</table>

**Applicability of Graphicrawl**

Consider the hypothetical situation in which a programmer is working on a website, any website. Say he is maintaining it, and hasn’t got much experience with the website. Below are several questions that the programmer may ask when trying to figure out how both the back-end and the front-end of the website works. Please indicate to what extent you would agree that Graphicrawl helps with answering each of these questions, ranging from 1 (completely disagree) to 5 (completely agree).

<table>
<thead>
<tr>
<th>Question</th>
<th>Agreement</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 What in this structure distinguishes these cases?</td>
<td></td>
</tr>
<tr>
<td>21 Under what circumstances is this method called or exception thrown?</td>
<td></td>
</tr>
<tr>
<td>22 How does the system behavior vary over these types or cases?</td>
<td></td>
</tr>
<tr>
<td>23 What will be (or has been) the direct impact of this change?</td>
<td></td>
</tr>
<tr>
<td>24 What will be the total impact of this change?</td>
<td></td>
</tr>
<tr>
<td>25 Will this completely solve the problem or provide the enhancement?</td>
<td></td>
</tr>
</tbody>
</table>