TreeTest: Online Tree Testing for Information Hierarchies

Ajdin Mehic

TreeTest: Online Tree Testing for Information Hierarchies

Ajdin Mehic

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 14 Oct 2019

TreeTest: Online Tree Test für Informationshierarchien

Ajdin Mehic

Masterarbeit

für den akademischen Grad

Diplom-Ingenieur

Masterstudium: Informatik

an der

Technischen Universität Graz

Begutachter

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 14 Oct 2019

Diese Arbeit ist in englischer Sprache verfasst.

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The document uploaded to TUGRAZonline is identical to the present thesis.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Dokument ist mit der vorliegenden Arbeit identisch.

Date/Datum

Signature/Unterschrift

Abstract

In the field of information architecture (IA), tree testing is a relatively new method for evaluating an information hierarchy. Representative test users are asked to locate items within a plain, bare-bones information hierarchy and their progress is recorded and analysed. Tree testing can be performed using paper cards, locally installed software, or online web applications.

TreeTest is an open-source fullstack JavaScript web application for tree testing, based on a MEAN stack (MongoDB, Express.js, Angular, Node). It provides the basic functionality required for creating, running, and analysing the results of a tree testing study. In future, it can serve as the basis for an open-source alternative to existing commercial tree testing applications.

Kurzfassung

Im Bereich der Informationsarchitektur (IA) ist das Tree Testing eine relativ neue Methode zur Bewertung einer Informationshierarchie. Repräsentative Testbenutzer werden gebeten, Elemente in einer einfachen Informationshierarchie zu lokalisieren und deren Fortschritte werden dabei aufgezeichnet und analysiert. Tree Testing kann mit Papierkarten, lokal installierter Software oder mit Online Web Applikationen durchgeführt werden.

TreeTest ist eine Open-Source-Fullstack-JavaScript Web Applikation zum Tree Testing, die auf einer MEAN Stack (MongoDB, Express.js, Angular, Node) basiert. Sie bietet die grundlegenden Funktionen zur Erstellung, Ausführung und Analyse der Ergebnisse einer Tree Testing Studie. In Zukunft kann sie als Grundlage für eine Open-Source-Alternative zu bestehenden kommerziellen Tree Testing Applikationen dienen.

Contents

List of Figures

List of Tables

List of Listings

Acknowledgements

I would like to thank my advisor, Keith Andrews, for all his support. His door was always opened for me, and he helped keeping me in the right direction. I wish to thank him for endless hours invested in correcting draft version of this thesis, and pointing out how I should implement, and improve TreeTest web application. It was not always easy (especially because I was working full-time in parallel), but looking at the final version of the thesis, and finished web application, I am extremelly happy and proud.

I would also like to thank my family and friends, who had to deal with me on stressful days.

Ajdin Mehic
Graz, Austria, 14 Oct 2019

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2019].
- Figure 2.1 was adapted by Keith Andrews from Figure 2-2 of Spencer [2014, page 28], and is used with kind permission of Keith Andrews.
- Figure 2.3 was adapted by Keith Andrews from Figure 26 of Schilb [2006, page 40], and is used with kind permission of Keith Andrews.
- Figure 3.1 was extracted from Borreguero Llorente et al. [2018, page 15] and is used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.
- Figure 3.2 was extracted from Borreguero Llorente et al. [2017, page 12] and is used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.
- Figure 3.3 was created by Keith Andrews and is used with kind permission.
- Figure 3.4 was extracted from Gaffney [2006], and is used with permission.
- Figure 3.5 was extracted from Gaffney [2006], and is used with permission.
- Figure 3.6 is extracted from Schilb [2006, page 56] and is used with kind permission of Steffen Schilb.
- Figure 3.7 is extracted from Schilb [2006, page 88] and is used with kind permission of Steffen Schilb.
- Figure 3.8 is extracted from Schilb [2006, page 61] and is used with kind permission of Steffen Schilb.
- Figure 3.9 is extracted from Schilb [2014] and is used with kind permission of Steffen Schilb.
- Figure 3.10 is extracted from Schilb [2014] and is used with kind permission of Steffen Schilb.

Chapter 1

Introduction

This thesis focuses on the tree testing technique for evaluating an information hierarchy. The first part of the thesis explains the ideas behind information architecture, information hierarchies, and tree testing. Tree testing strategies and existing tools are surveyed. To lay the ground for developing an online tree testing tool, some of the fundamental modern web technologies are presented.

The second part of the thesis presents TreeTest, a fullstack JavaScript web application for tree testing based on a MEAN stack (MongoDB, Express, Angular, Node). TreeTest implements the basic functionality required for creating, running, and analysing the results of a tree testing study. A formative user study (thinking aloud test) was conducted to discover potential usability issues in the TreeTest interface, which were subsequently addressed.

The thesis concludes with three appendices, each containing a user guide for a particular kind of TreeTest user. Appendix A contains the Administrator Guide for users who install and maintain a TreeTest server. Appendix B contains the Study Owner Guide for users who intend to create and run a tree testing study. Appendix C contains the Participant Guide for end users who participate in a tree test.

Chapter 2

Information Architecture

Information architecture (IA) is the structural design of an information space to facilitate intuitive access to its contents [Rosenfeld et al. 2015, Chapter 2]. IA is everywhere. It is in websites, manuals, and smartphone apps for workflows, menu structures, and navigation. It even underlies the organisation of some physical places. A good IA helps users understand their surroundings and find what they are looking for [Spencer 2014, page 28]. IA includes groups and subgroups (hierarchical structure) for content and the labels used at each level [Martin 2019, Chapter 3]. Where collections of homogeneous items (for example, products) are present, IA also includes the various metadata fields used to characterise the items. An example of where IA is involved in a product website is shown in Figure 2.1.

Consider the following situation. A user opens a website for the first time and their initial impression based on the website's landing page is of an aesthetic design and good overview. The user wants to find some particular information, but the site navigation does not suggest where that information might be located. Websites often have well-designed visuals. They can look fancy and have well-written content. At the same time, the underlying information structure can be illogical [Schilb 2006, Chapter 2]. Users are unable to find what they are looking for, even if it is actually present. This can result in frustration and, eventually, in users going elsewhere.

2.1 Building an Information Hierarchy

An information hierarchy can and should be prototyped on paper, using post-it notes or index cards, well before even starting to build the corresponding user interface or application, as shown in Figure 2.2. Solving structural issues on a live website or application is possible, but can be time-consuming and complicated. Users might not be happy seeing a new and different design and this can have a negative impact on returning users, who are already familiar with the site structure.

One of the first steps in building an information hierarchy is often *card sorting*. Card sorting [Spencer 2009; Rosenfeld et al. 2015, pages 166–168] is a quick and easy way to design an information hierarchy. Users are asked to first group concepts together, then label the groups, as illustrated in Figure 2.3. Card sorting can help discover how people think content should be organised and discover insights from a user perspective. Generally, the output of card sorting is a grouping for items at one level of a hierarchy, as well as suggested labels for each group. Card sorting proceeds in a bottom-up fashion, progressively grouping the labels from a lower level, to construct an initial information hierarchy.

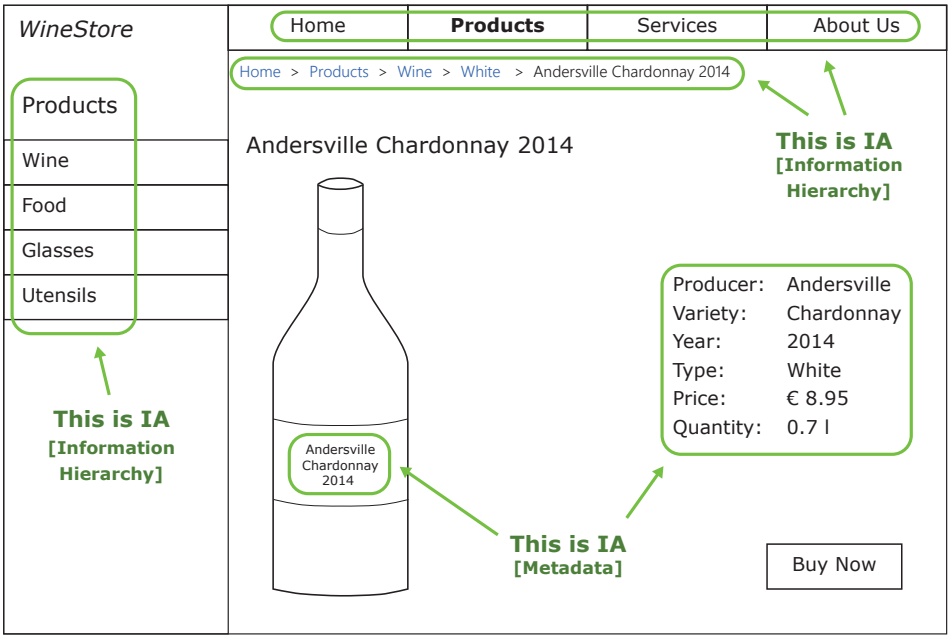


Figure 2.1: An example of where information architecture is involved in a website selling various products. [Adapted by Keith Andrews from Figure 2-2 of Spencer [2014, page 28], and used with kind permission of Keith Andrews.]

Figure 2.2: An information hierarchy should be prototyped using post-it notes or index cards, well before even starting to build the corresponding user interface or application.

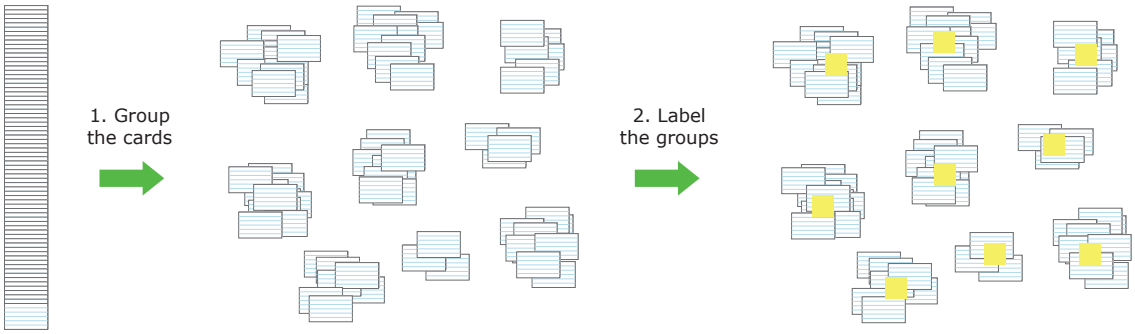


Figure 2.3: In card sorting, users are asked to first group concepts together, then label the groups. [Adapted by Keith Andrews from Figure 26 of Schilb [2006, page 40], and used with kind permission of Keith Andrews.]

2.2 Testing an Information Hierarchy (Tree Testing)

Once an initial information hierarchy has been designed, it has to be tested with representative users from the target population. One relatively new method for evaluating an information hierarchy is *tree testing*. Tree testing is a usability technique, where representative test users are asked to locate items within a plain, bare-bones information hierarchy and their progress is recorded and analysed [O'Brien 2017a]. Tree testing is also known as reverse card sorting or card-based classification.

For example, a potential information hierarchy might be under consideration for a website. It contains the items "Home", "Products", "Services", and "About Us". Each of these items contains a subhierarchy. The question is whether the current information hierarchy is logical and if it will make sense for users. Therefore, users are given tasks such as:

- Where would you expect to find food products?
- Where would you expect to find wine products?
- Where would you expect to find contact e-mail?
- Where would you expect to find all services?

The results of these tests can give an insight into whether the site's information hierarchy is working. After some analysis, it is often possible to reorganise the information hierarchy to give users a better and more logical view.

A participant in a tree test study typically follows a procedure like this:

1. The participant is given a "find it" task, such as "Where would you expect to find wine products", starting from the top of the tree.
2. The participant sees a list of top-level topics of the website.
3. The participant chooses a topic and then sees a list of subtopics.
4. The participant continues navigating (moving down the tree) until the topic satisfying the task is found (or the participant gives up).

Once a certain number of participants have completed the test, the results are analysed by considering questions such as these:

- Were users successful in finding a particular item in the tree?

- Was the item found directly, or did users need to back up?
- If the item was not found, where did users go instead?
- How much time did users need for each task?
- Overall, which parts of the tree worked well and which did not?

By answering these questions, it is usually possible to redesign and improve the website's information hierarchy.

Tree testing has the following advantages:

- Sessions are relatively short, which makes recruitment of participants easier.
- Testing can be done face-to-face or remotely.
- Analysing the data can be done quickly and results can be acted upon quickly.

Of course, there are some disadvantages:

- The information hierarchy is deliberately shown in a basic form without visual elements or decoration. The experience with a real interface will be different.
- When tree testing is done remotely, researchers are not able to observe the test and pay attention to how users made decisions and why they made those choices.

Tree testing was originally performed manually with paper cards, but now it is possible to use specialised software such as Treejack [OW 2019] and UserZoom [UZ 2019]. These are described in more detail in Chapter 3.

One example of the use of tree testing is described by Le et al. [2014] in the field of Health Information Technologies (HIT). These systems include various applications such as electronic health records, electronic prescription systems, and computerised provider order entry systems. As Le et al. [2014] discuss, systems with poor usability have been found to negatively impact patient mortality, while well-designed user-tested systems can have a positive impact on patient safety.

Chapter 3

Tree Testing Strategies and Tools

A first attempt to create an information hierarchy will seldom be perfect. It is usually necessary to test and refine an information hierarchy through a series of iterations of tree testing with representative end users.

3.1 Paper-Based Tree Testing

In paper-based tree testing, users are asked to talk out loud while navigating an information hierarchy which has been implemented on paper cards [Gaffney 2001]. Figure 3.1 shows the entire set of cards representing the information hierarchy of Graz University of Technology's external website [TUG 2019a], containing 686 nodes upto 6 levels deep [Borreguero Llorente et al. 2018]. Figure 3.2 shows a participant doing a task in the paper-based tree test [Borreguero Llorente et al. 2017] of the information hierarchy of Graz University of Technology's intranet site [TUG 2019b], which has 483 nodes upto 5 levels deep. Each time the participant chooses a particular item, the facilitator or an assistant places cards representing the children of that item onto the table for the next level of choices.

Paper-based tree testing has the advantage that participants are face-to-face, giving the facilitator the flexibility to explore any issues as they occur and to gain insight into the user's thought process. However, modelling an entire hierarchy on paper cards is rather cumbersome. It is also tricky to locate and present the next level of the tree structure interactively in response to a user's choice. The entire test procedure is quite time-consuming and the chosen paths and deviations of each user have to be recorded manually (or later extracted from a video recording).

3.2 Locally Installed Software

A tree test can be performed using locally installed software, where users are asked to do a tree test sat at a PC where the software is installed. Internet access is not necessary, since the results are saved on the same PC, or manually by the facilitator sitting next to the test user.

3.2.1 Folders and Subfolders

An information hierarchy can be modelled using folders and subfolders in the file system, as shown in Figure 3.3a. Test users are then asked to navigate through the folder hierarchy to indicate where they would expect to find certain items, while thinking out loud. During such a tree test, the user is typically only shown one level of the hierarchy at a time, as shown in Figure 3.3b. Brian Hoffman describes such a procedure in Spencer [2014, page 327].

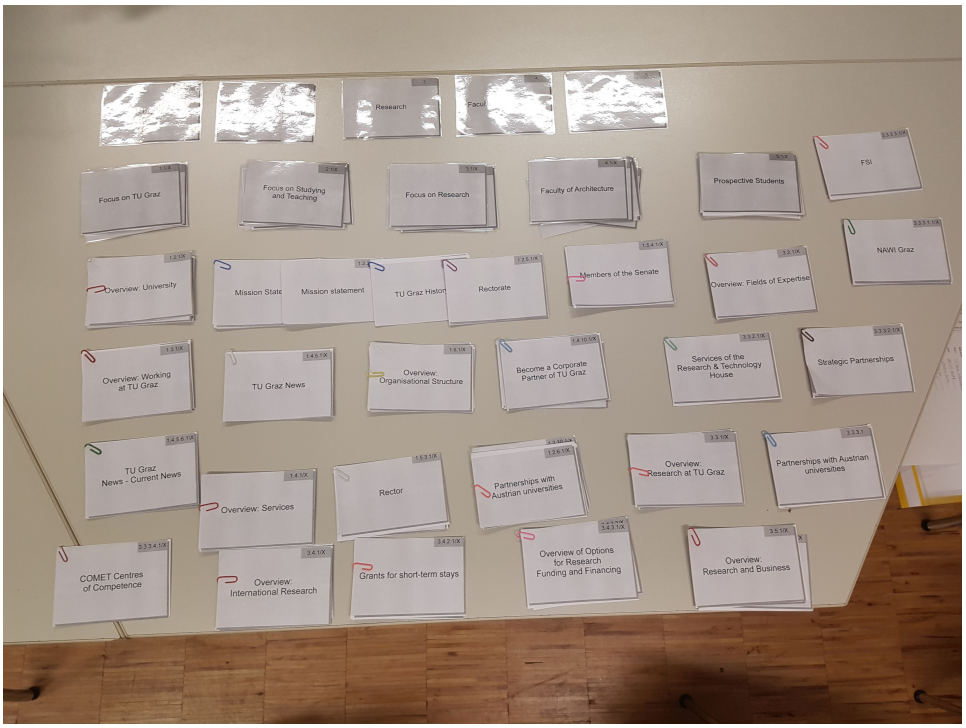
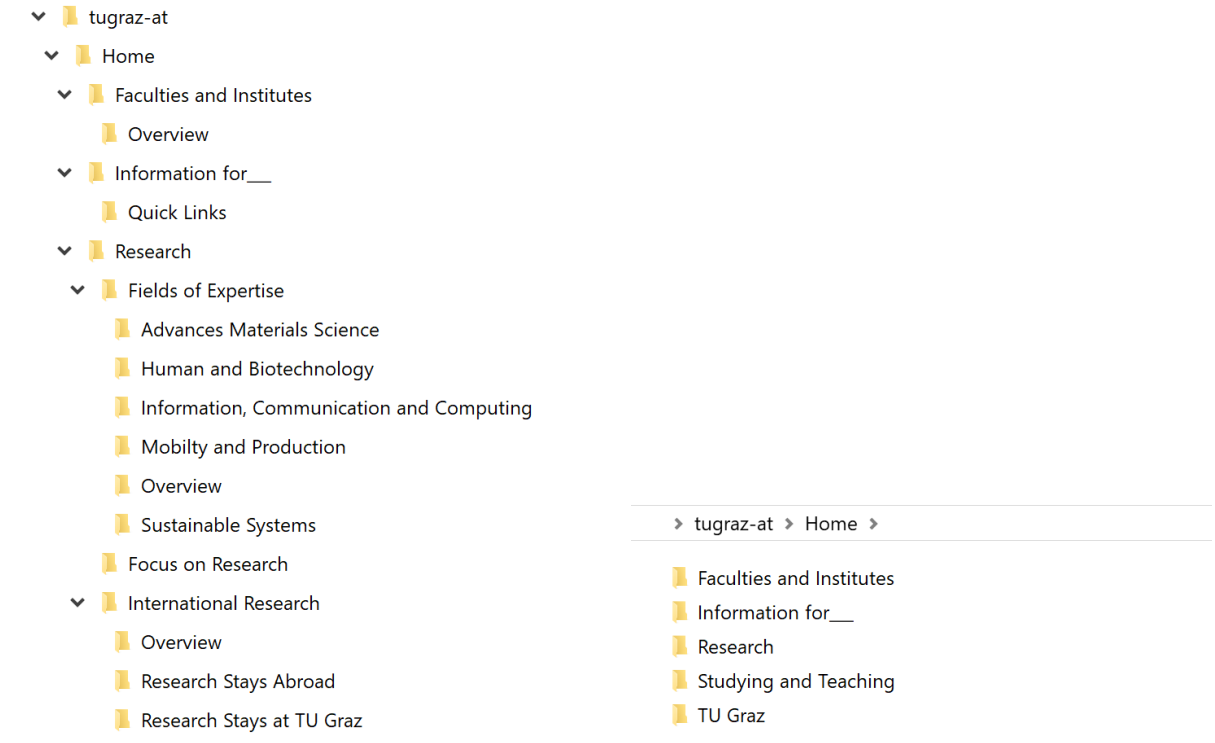


Figure 3.1: An entire tree structure modelled using cards for paper-based tree testing [Image extracted from Borreguero Llorente et al. [2018, page 15] and used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.].

Figure 3.2: A participant doing a task in a paper-based tree test [Image extracted from Borreguero Llorente et al. [2017, page 12] and used under the terms of the Creative Commons Attribution 4.0 International (CC BY 4.0) licence.].



(a) Part of an information hierarchy for a university website, modelled as folders and subfolders in the file system.

(b) During a tree test, a user is presented with one level at a time in the file system's explorer.

Figure 3.3: Tree testing with folders and subfolders. [Image created by Keith Andrews and used with kind permission.]

3.2.2 Classified

Classified was installable application which ran under Windows 95 and Windows 98. It was "designed to evaluate navigational structures for web sites, documents, program menus, catalogues and other information spaces." [Gaffney 2006]. Users were asked in which of upto 20 categories they would expect to find certain items. However, subcategories were not supported, so only one level of a hierarchy could be modelled and tested at a time. The results were saved locally in a CSV file.

The workflow for a classification test study was as follows:

1. Specify the categories and items. A text file was edited to specify upto 20 categories and the items they each contained. Categories are enclosed in square brackets, followed by items belonging to the category, as shown in Listing 3.1. The classification file was then imported into the tool.

2. Perform a series of tests. The study owner could enter a number for each participant, as shown in Figure 3.4, and start the test. Target items were randomly selected, and the test user had to select the most appropriate category, as shown in Figure 3.5. The test user could make upto three attempts to locating each item.

3. Analyse the results. The results were saved locally as a CSV for further analysis. For each user and item, the actual location, and first, second, and third attempts were recorded, as shown in Table 3.1.

```
[Contact Us]
Getting Here
Telephone Numbers

[Services]
Telephone Support
Consulting

[Downloads]
User Guides
Tutorials
Updates

[Utilities]
Conversion Scripts
Output to CSV

...

[end]
```

Listing 3.1: Classified. Part of a classification file used to specify categories and the items each category contains.

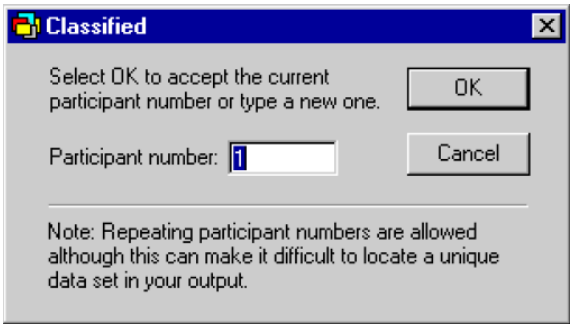


Figure 3.4: Classified. The facilitator starting a test. [Image extracted from Gaffney [2006], and used with permission.]

Participant	Item	Actual Location	First Attempt	Second Attempt	Third Attempt
1	Telephone Numbers	Contact Us	FAQ	Contact Us	
1	Tutorials	Downloads	Services	Reading List	Downloads
2	Telephone Numbers	Contact Us	Contact Us		
2	Tutorials	Downloads	Guided Tour	Downloads	

Table 3.1: Classified. The results are saved as a table in CSV format.

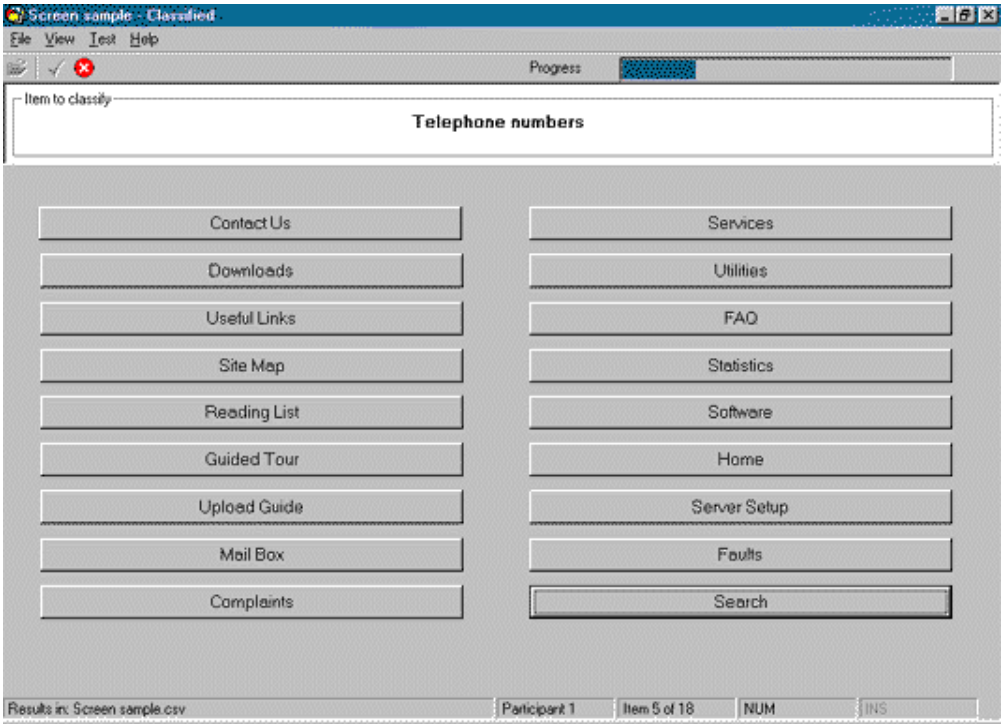


Figure 3.5: Classified. A user's view of a test. [Image extracted from Gaffney [2006], and used with permission.]

3.3 Online Tree Testing Tools

Online tools for tree testing provide an infrastructure for the developer of an information hierarchy to efficiently test it. The tree structure is uploaded and test users can complete navigation tasks online with a web browser. The information hierarchy is presented to test users as a plain, bare-bones hierarchy without visual embellishments. Many users can be invited to participate and testing can be done remotely and unsupervised, or face-to-face. The advantage of face-to-face testing is that a study owner can ask test users why they made a certain decision [Spencer 2014, page 318]. This helps understanding why information architecture is working (or not working). The resulting click paths and timings are collected automatically and the results are displayed graphically and made available for download.

3.3.1 Heurekia

Heurekia was an online tool for testing website navigation hierarchies [Schilb 2006, Chapter 4]. The study owner could enter the information hierarchy, test tasks, and configure an introductory message. An example of creating tasks for a study is shown in Figure 3.6. The study owner could enter the list of test users, which included first name, and e-mail address. After publishing the study, each of the test users was sent an e-mail with the test link. An example of a user performing a test is shown in Figure 3.7. After a sufficient number of users had performed the test, the study owner could view the results, as shown in Figure 3.8.

3.3.2 C-Inspector

C–Inspector [Schilb 2014] was a web–based application which helped test an information hierarchy, and was created in May 2009 as a follow-up to Heurekia. It was closed on 01 Jan 2014. In C-Inspector, the study procedure comprised three steps: prepare a study, run a study, and analyse the study results.

While creating a study, one or multiple answers per task could be defined. It was possible to enter the number of attempts per task (1–5 or unlimited), as well as choosing the study language (English or German). It was also possible to configure text for welcome, instructions, and thank you messages.

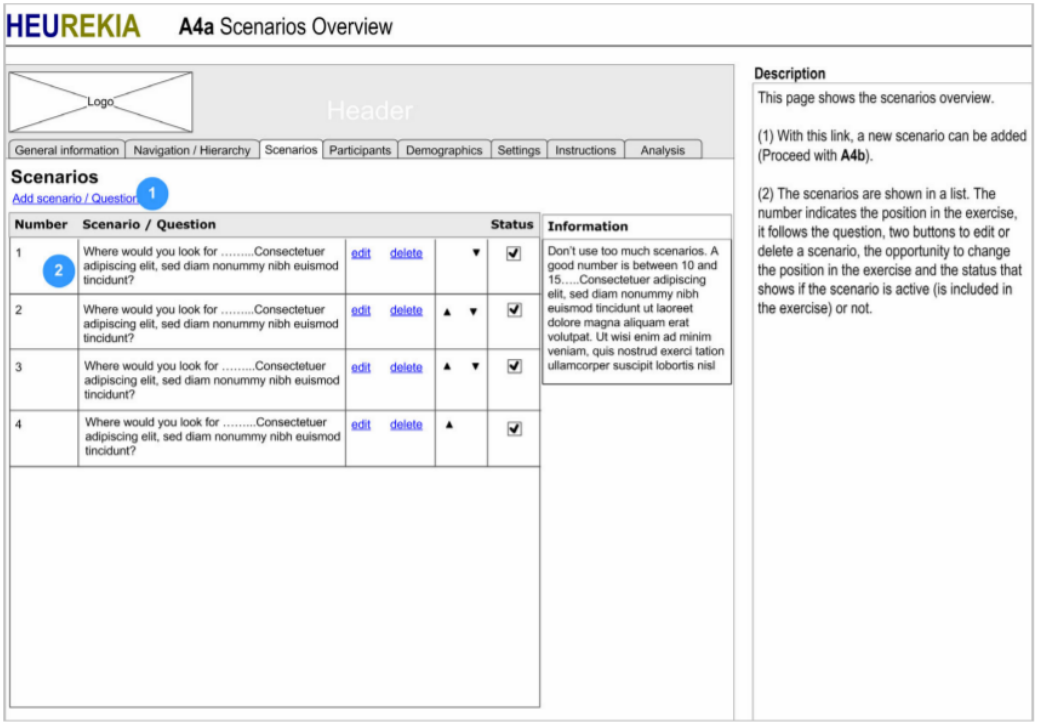


Figure 3.6: Heurekia. Defining tasks. [Image extracted from Schilb [2006, page 56] and used with kind permission of Steffen Schilb].

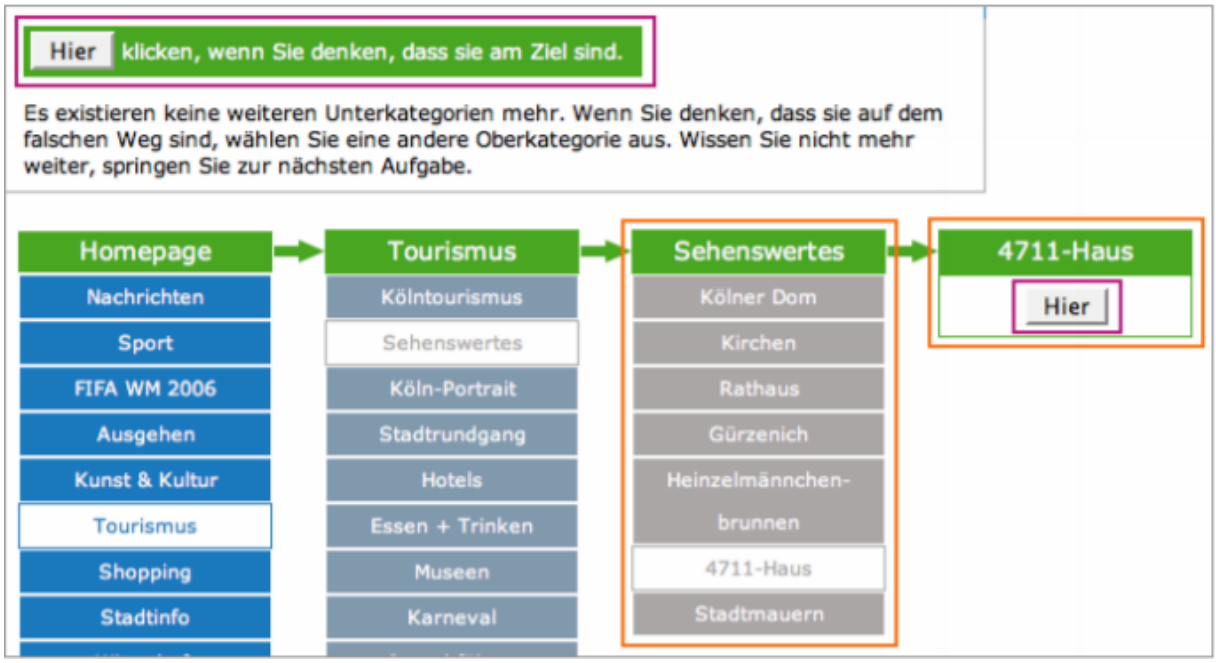


Figure 3.7: Heurekia. User's view of a test. [Image extracted from Schilb [2006, page 88] and used with kind permission of Steffen Schilb].

Figure 3.8: Heurekia. Task statistics. [Image extracted from Schilb [2006, page 61] and used with kind permission of Steffen Schilb].

An example of a participant doing a study in C-Inspector is shown in Figure 3.9. After a sufficient number of participants had completed the tree test, the results by task and participant could be seen on the result analysis page, shown in Figure 3.10.

3.3.3 PlainFrame

PlainFrame was an online tool for remote usability testing of website navigation, built using Flash [PlainFrame 2014]. First, a new study had to be created, consisting of a sitemap, and a set of navigation tasks. The study link was then sent to potential test participants. After a sufficient number of users had performed the test, the owner of the study could view various results, including charts, statistics, records of every user interaction, and the option to replay any user session. On 25 Feb 2014, Optimal Workshop's CEO, Andrew Mayfield, announced that Optimal Workshop has acquired PlainFrame. PlainFrame users were migrated to the Optimal Workshop platform. PlainFrame itself was shut down on 03 Mar 2014.

3.3.4 Naview

Naview was a navigation preview tool for rapid information architecture prototyping [Volkside 2019]. The Naview paid service was discontinued at the end of March 2017, but the limited free plan is still operational. Naview helped information architects design and visualise a new navigational structure, bridging the gap between card sorting and IA user testing. In order to start creating tests in Naview, one had to create a free account. Then it was necessary to create a study and enter a tree. Entering a tree was possible either by pasting or entering the tree structure manually. Different levels of tree were identified by tabs, as shown in Figure 3.11. After that, it was possible to start writing tasks, as shown in Figure 3.12, and eventually publish the study.

When doing a study, each user is asked to navigate through the tree until the entry satisfying the task is found, as shown in Figure 3.13. It is also possible for the user to skip a task, if it was not possible to find the corresponding item.

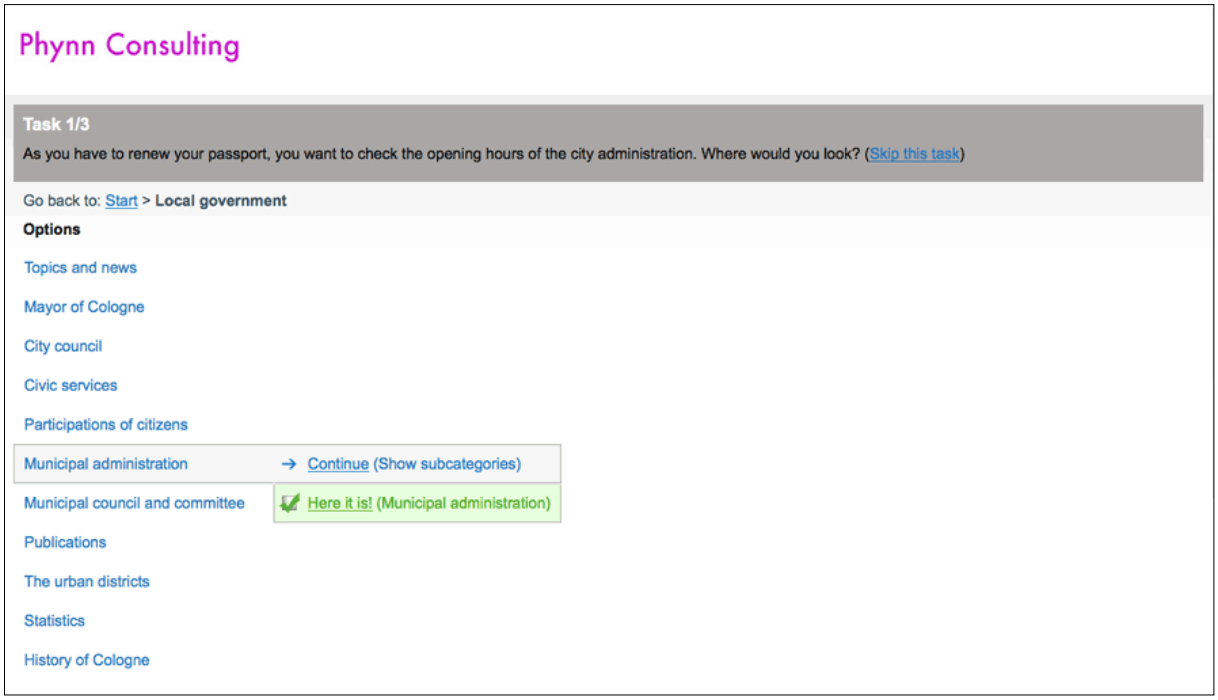


Figure 3.9: C-Inspector. User navigating through a tree to find the correct answer. [Image extracted from Schilb [2014] and used with kind permission of Steffen Schilb].

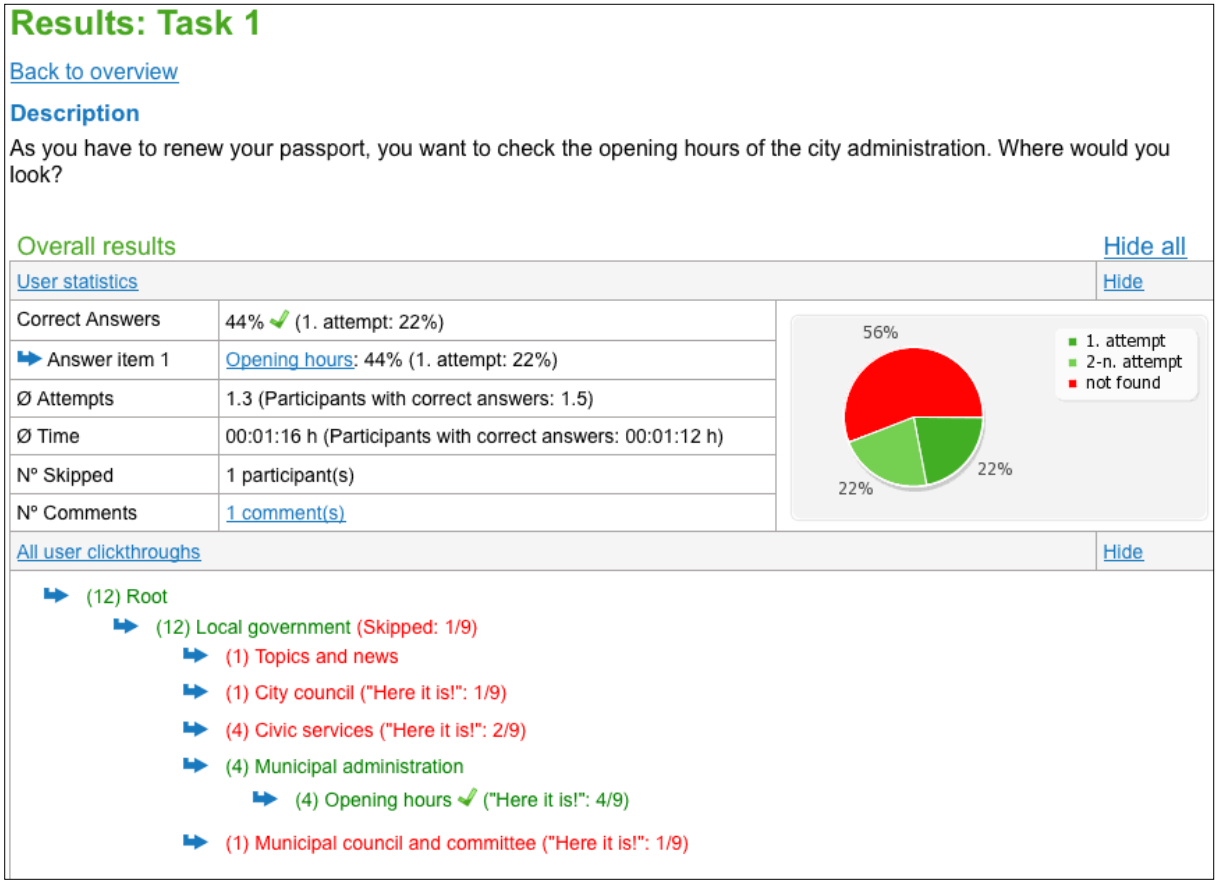


Figure 3.10: Result analysis in C-Inspector [Image extracted from Schilb [2014] and used with kind permission of Steffen Schilb].

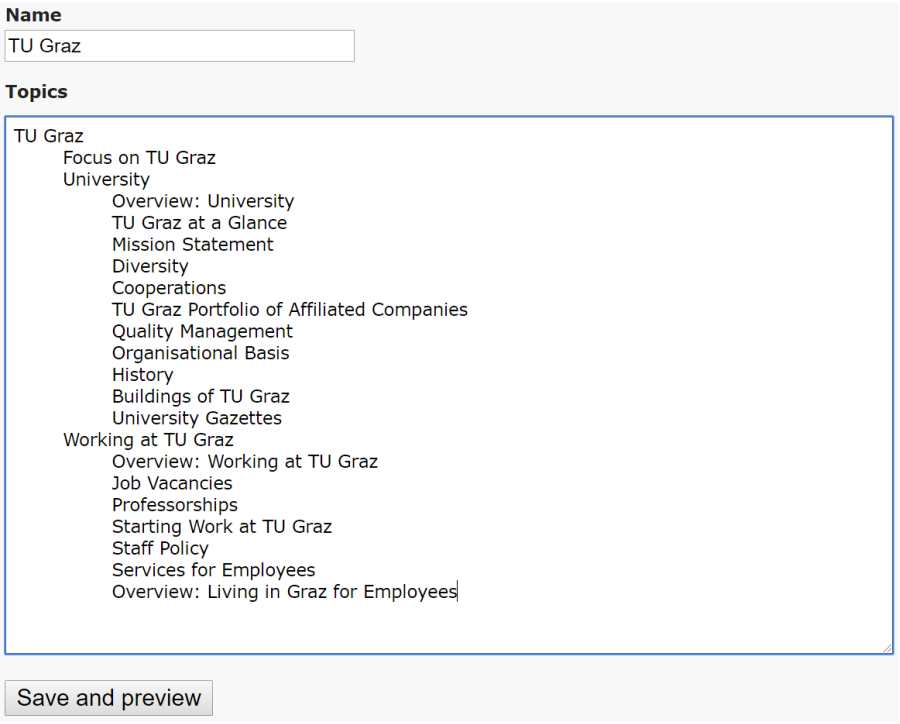


Figure 3.11: Naview. Entering the tree structure manually, different tree levels are identified by tabs. [Screenshot of Naview, made by the author of this thesis].

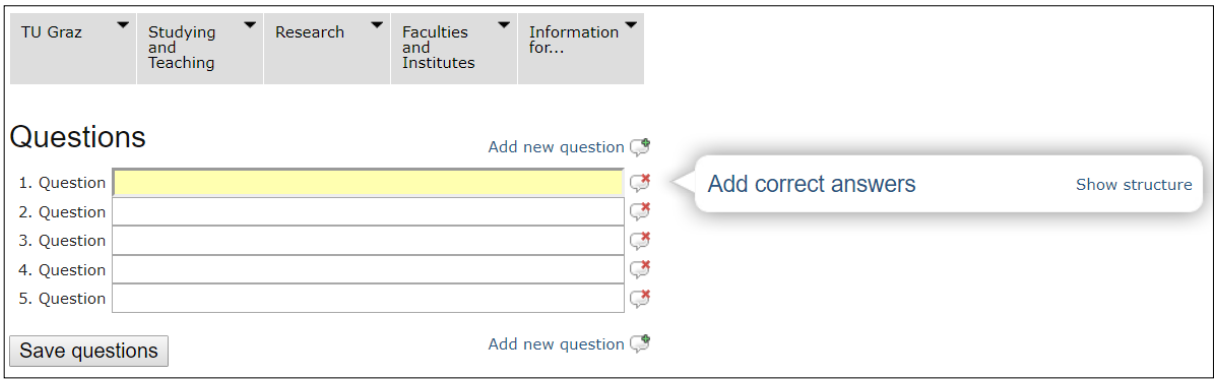


Figure 3.12: Naview. Defining tasks. [Screenshot of Naview, made by the author of this thesis]

Naview study results contain information about the total number of participants in the study, and overall success rate. For each task, individual and average success rate, completion time, and directness are displayed. Clicking on a task shows a summary overall of all clicks by all users for that task, as illustrated in Figure 3.14.

3.3.5 UserZoom

UserZoom is a platform for testing and measuring the user experience of websites, apps, and prototypes [UZ 2019]. Among many other tools, it also provides a tree testing tool. Creating a tree test project in UserZoom is done through the following steps:

1. Definition. Within the definition section, the study owner first has to choose a device type. It can be desktop, mobile, or both at the same time. Next, the basic information about the study (title, description, and tree structure) and the participants' language is specified. Figure 3.15 shows the

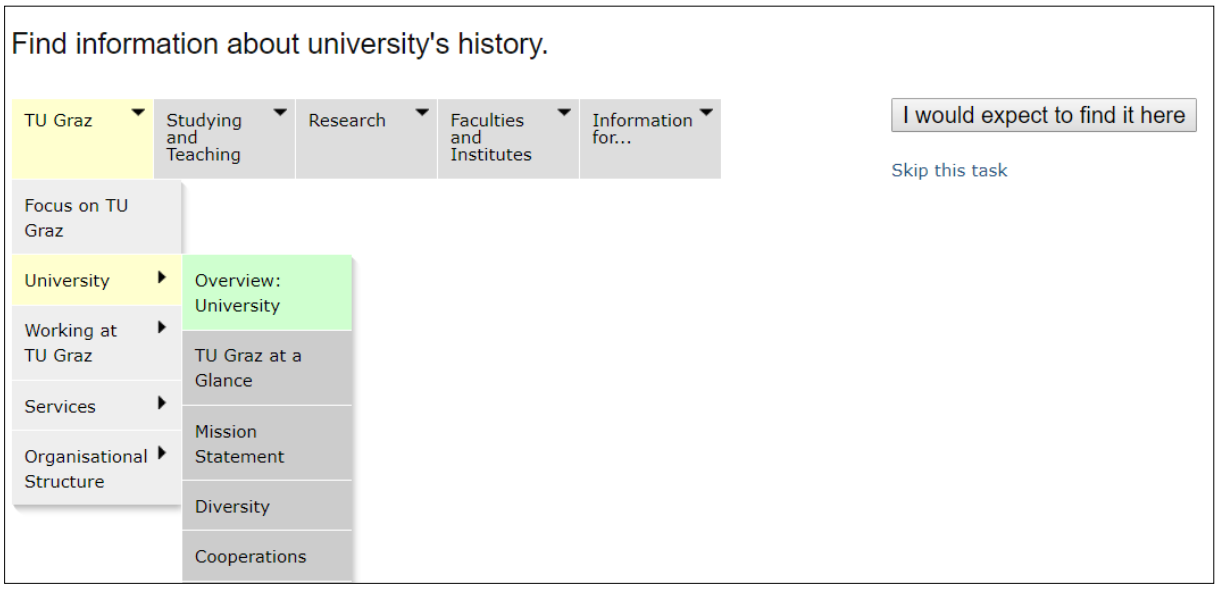


Figure 3.13: Naview. A user navigating through the tree to find the correct answer. [Screenshot of Naview, made by the author of this thesis]

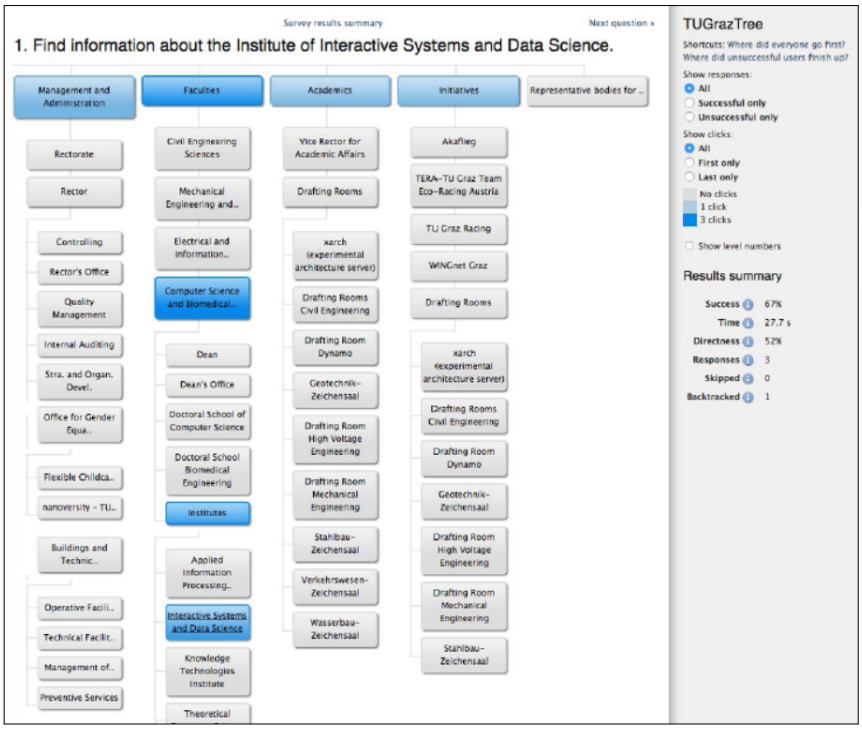


Figure 3.14: Naview. Task statistics. [Image extracted from Borreguero Llorente et al. [2017, page 20] and used under Creative Commons Attribution 4.0 International (CC BY 4.0) licence.].

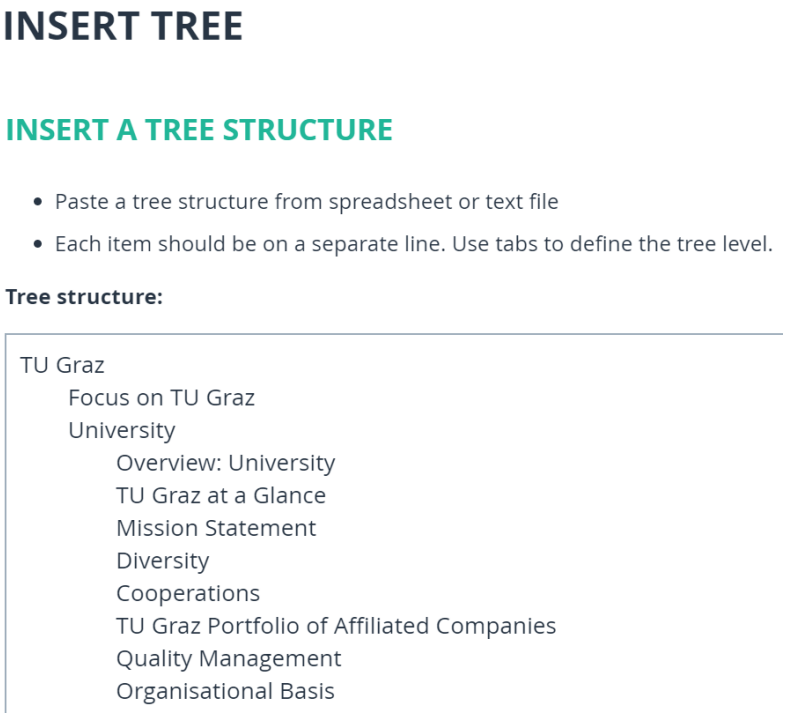


Figure 3.15: UserZoom. Entering the tree structure manually. Different tree levels are specified by tabs. [Screenshot of UserZoom, made by the author of this thesis].

tree structure being entered.

2. Participants. Under certain paid plans, there is the possibility to ask UserZoom to select and provide participants, based on specified criteria. Otherwise, participants can be invited by email, social media, or by a QR code, as shown in Figure 3.16.

3. Task builder. The study owner defines tasks and corresponding answers in the tree structure, as shown in Figure 3.17.

After a tree test is published and shared, participants can access it. Figure 3.18 shows an example of a participant doing a task. Once a sufficient number of users have done the test, the owner of the study can view statistics, including an overview of participants, task clicks analysis, and general information about operating systems, browsers, and screen resolution used by the participants. Figure 3.19 shows an example of task clicks analysis.

3.3.6 Treejack

Treejack is an online tree testing tool by Optimal Workshop [OW 2019]. The owner of the study first creates a new tree testing study and optionally customises the default welcome text. Next, the information hierarchy can be imported from a file, or can be entered manually into Treejack's interface, as shown in Figure 3.20. When the tree is ready, it is possible to define a background questionnaire, feedback questionnaire, and a set of tasks. Each task asks the user to find a requested item in the tree, as can be seen in Figure 3.21.

Once the tree structure and tasks have been setup, Treejack generates a unique study link (URL), which can be sent by email to potential participants. When a user opens the link, the set of questions are presented one after another, and the user tries to find the requested items in the tree, as shown in Figure 3.22. Treejack logs all user interactions for later analysis.

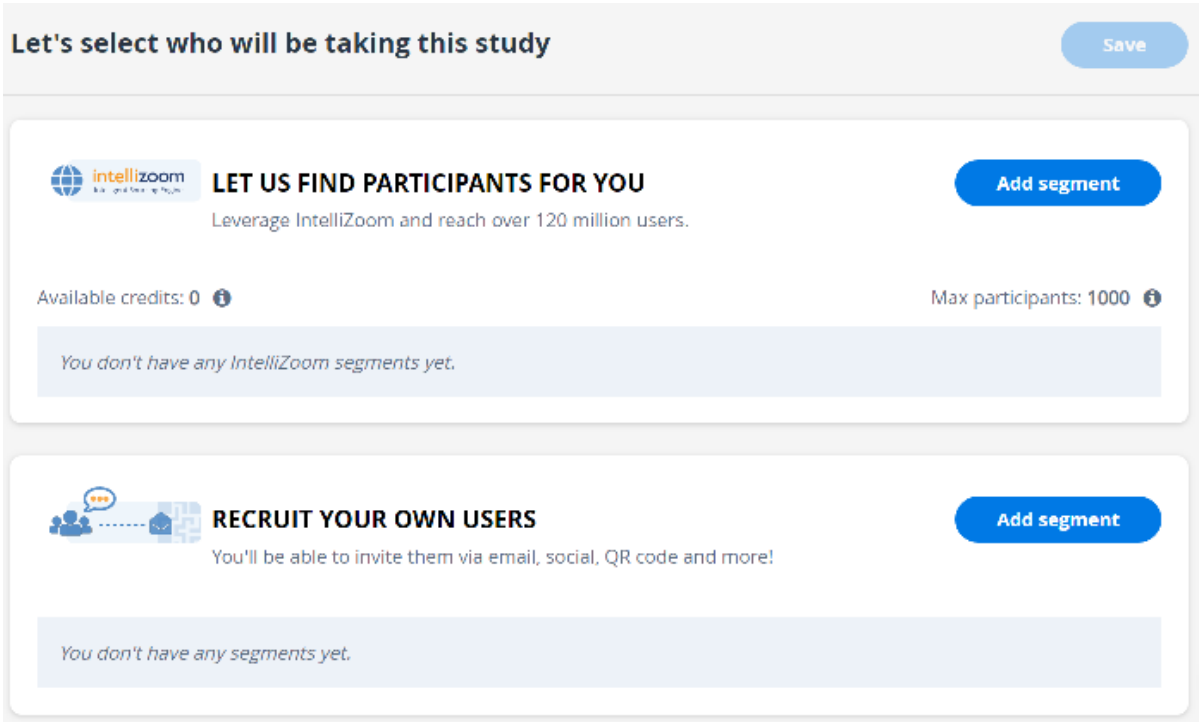


Figure 3.16: UserZoom. In some paid plans, the study owner can ask UserZoom to select participants based on specified criteria. Otherwise, users can be invited by email, social media, or by a QR code. [Screenshot of UserZoom, made by the author of this thesis]

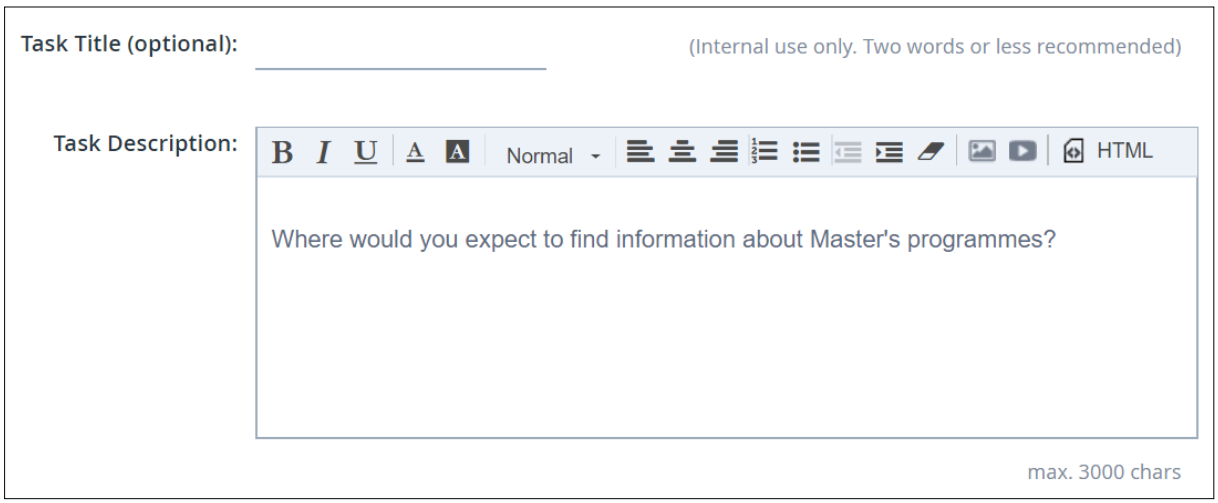


Figure 3.17: UserZoom. Defining tasks. [Screenshot of UserZoom, made by the author of this thesis].

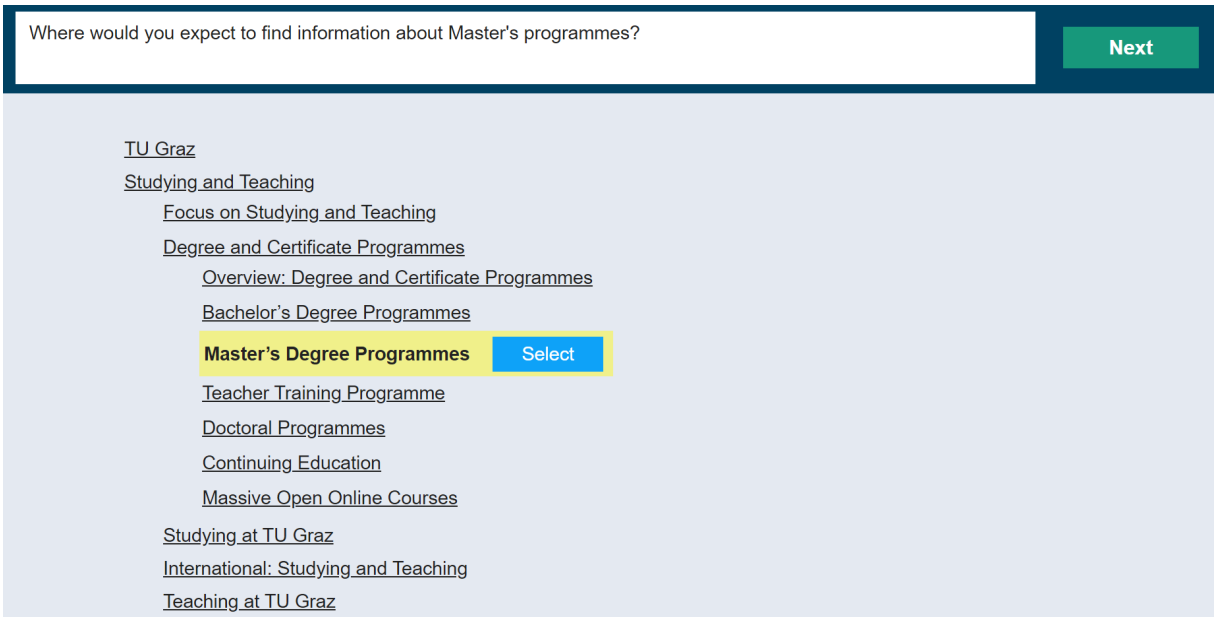


Figure 3.18: UserZoom. A participant is navigating through the tree to find the correct answer. [Screenshot of UserZoom, made by the author of this thesis]

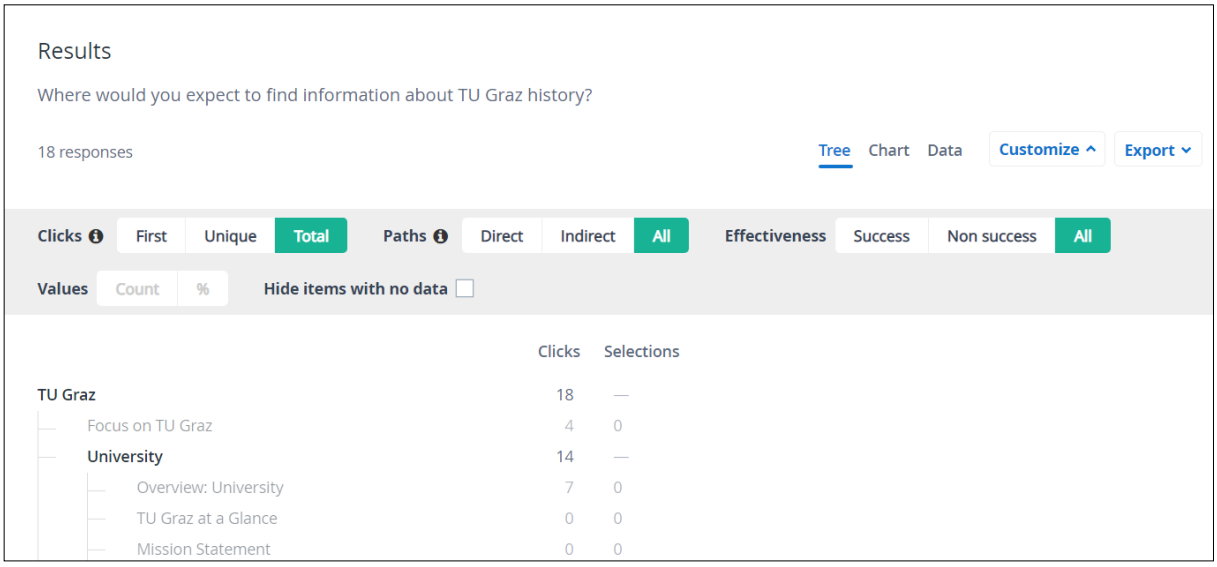


Figure 3.19: UserZoom. Task Statistics. [Screenshot of UserZoom, made by the author of this thesis]

Figure 3.20: Treejack. Entering the tree structure. [Screenshot of Treejack, made by the author of this thesis].

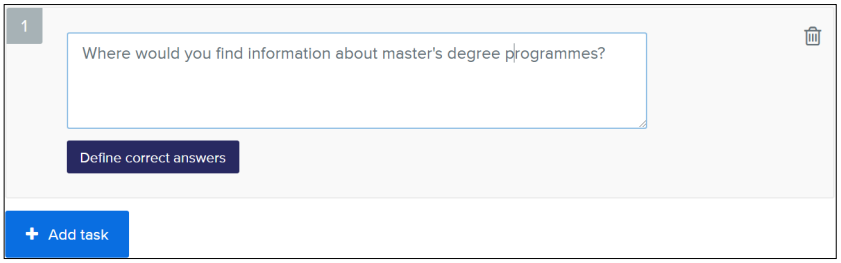


Figure 3.21: Treejack. Defining the tasks. [Screenshot of Treejack, made by the author of this thesis].

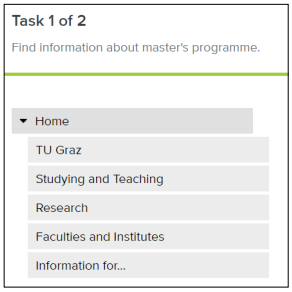


Figure 3.22: Treejack. A user's view of a test [Screenshot of Treejack, made by the author of this thesis].

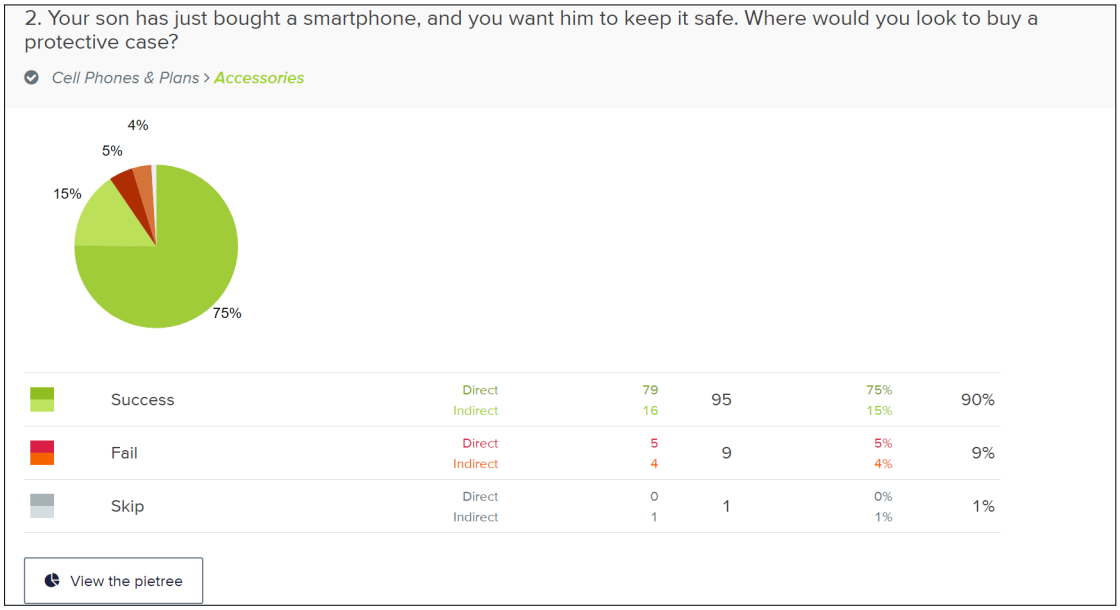


Figure 3.23: Treejack. Task statistics show how many participants found the correct answer for each task, how directly they arrived at their chosen answer, and how long they took to do it [Screenshot of Treejack, made by the author of this thesis].

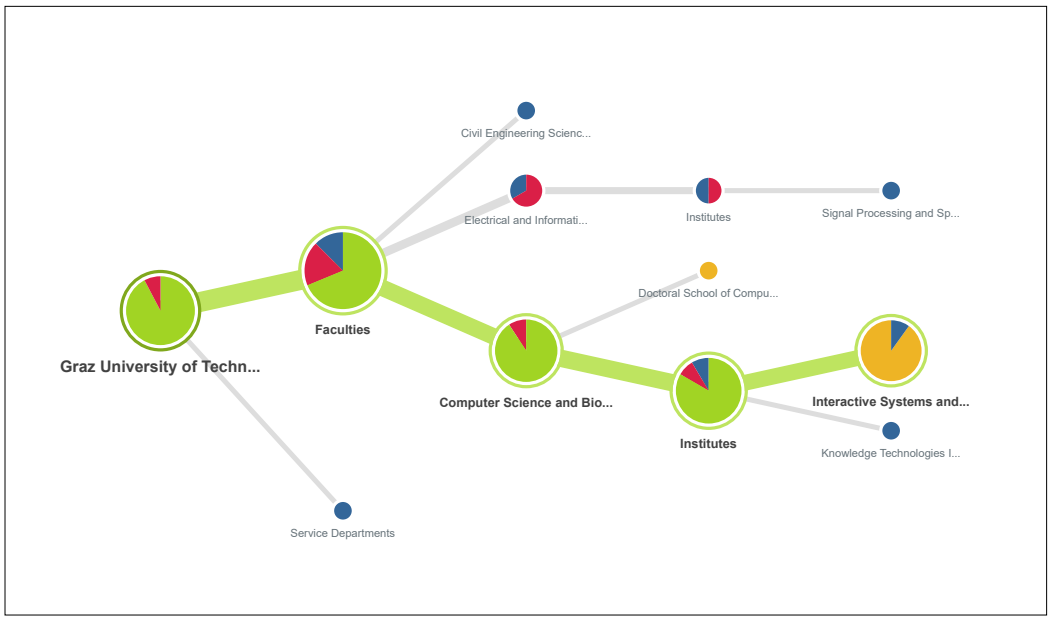


Figure 3.24: Treejack. The path analysis tool uses a pie tree to visualise the paths participants took while searching for the answer to a task. [Screenshot of Treejack, made by the author of this thesis].

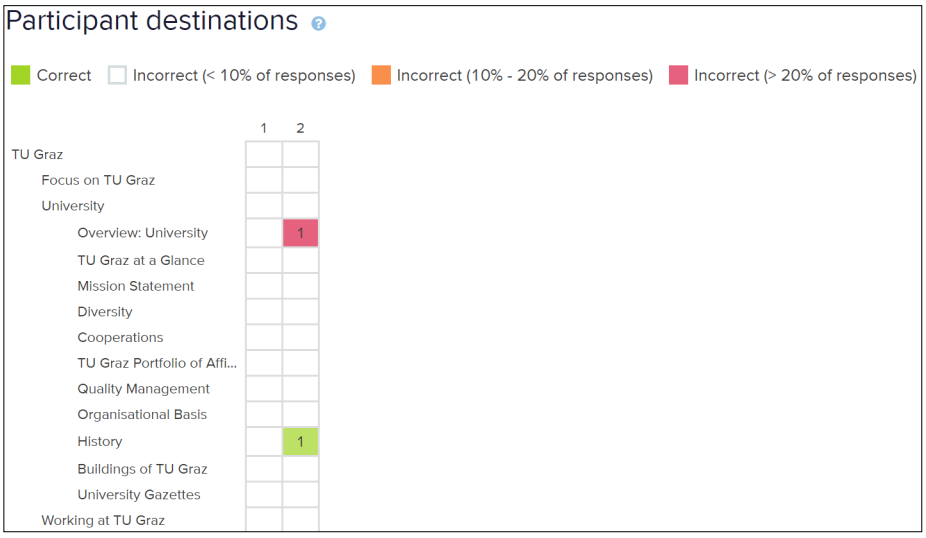


Figure 3.25: Treejack. The destinations table shows the number of clicks on each answer for each task [Screenshot of Treejack, made by the author of this thesis].

Once a sufficient number of users have done the test, the owner of the study can view various statistics:

- Task Statistics: The task statistics show how many participants found the correct answer for each task, how directly they arrived at their chosen answer, and how long they took to do it. This is illustrated in Figure 3.23.
- Path Analysis: Uses a pie tree (pietree) to visualise the paths participants took while searching for the answer to a task. An example is shown in Figure 3.24.
- Destinations Table: The destinations table shows number of clicks on each answer for each task. This is shown in Figure 3.25.

Treejack offers a free plan, in which it is possible to create a study with a maximum of 3 questions and 10 participants. In order to use the tool for a real study with an unlimited number of questions and users, a subscription to "The Suite" plan costs $166 a month. Students can obtain free access to this plan for a maximum of three months.

3.4 Analysis of Results

After a sufficient number of participants have done a tree testing study, the study owner has a set of valuable data for analysis. A number of quantitative measurements can be retrieved from tree testing for analysis, including task success rate, task directness rate, and task completion time.

3.4.1 Task Success Rate

The *success rate* for a task indicates the percentage of users who eventually (possibly with deviations from the direct path) managed to find where in the tree the correct answer is located. Selected locations other than the correct ones are reported as failures. In order to calculate the success rate, at least one correct answer needs to be assigned for each task. For example, if the task was to find "News from Austria" and 60 out of 100 participants eventually selected the correct location, the success rate for that task would be 60%. Overall success rate is a combined measure calculated over all of the tasks.

3.4.2 Task Directness Rate

The *directness rate* measures how many users selected the path to the correct answer without deviations, i.e. without navigating back in the tree in order to reconsider a decision. Backtracking typically occurs when a user navigates to some part of the tree and then does not find the desired answer. If a task has a high success rate, it does not necessarily mean the user experience was good. It can happen that many users must retrace their steps and try multiple paths before eventually finding the right answer.

3.4.3 Task Completion Time

Task *completion time* assesses how long users take to find the correct answer in the tree. The success rate can be high, but if it took too long for users to find the correct answers, it means that the overall information hierarchy may not be effective.

All three factors (success rate, directness rate, and completion time) should be taken into an account when investigating the effectiveness of the information hierarchy. O'Brien [2017b, Chapter 12] describes the analysis of results in more detail.

3.5 Comparison

Tree testing can be performed in a variety of ways with a variety of tools. Two choices stand out in particular:

- face-to-face or unsupervised.
- paper-based or computer-based (local or online).

The choices are orthogonal. For example, a computer-based study can use an online tool, but can still be run face-to-face with a facilitator.

Of the online tree testing tools presented in this chapter, all of them provide for study creation, online testing of a tree, and analysis of results. Treejack and UserZoom are among the 60,000 most visited websites in the world. UserZoom is mostly aimed at larger organisations, while Treejack offers services for individuals. Heurekia, PlainFrame, Naview, and C-Inspector have all been discontinued.

Chapter 4

Modern Web Technologies

The HyperText Markup Language (HTML) [Keith and Andrew 2016, Chapter 1], Cascading Style Sheets (CSS) [Cederholm 2014, Chapter 1] and JavaScript (JS) [Marquis 2016] are the three main and most important languages when it comes to web development [Robbins 2018]. HTML is used for specifying the content of the web page, CSS styles the content, while JS can make static websites interactive [Marquis 2016]. These programming languages provide the functionality to build modern, complex, and dynamic websites. However, using the plain web technologies, one might well end up with unmaintainable code, and consequent technical debt.

As web development has evolved, new technologies have been introduced which make development faster, safer, and more productive. These technologies include various JS frameworks, and programming languages for handling the server-side of a website. In this chapter, some of the most popular modern web technologies will be discussed.

4.1 Client-Server Model on the Web

On the web, the client-server model is used to regulate communication between a client (requester) and server (responder) over the internet, as shown in Figure 4.1:

- The web server runs continuously waiting for requests.
- The web client (browser) sends a request to the server.
- The server responds with an appropriate response.
- The client receives the response, parses it, and displays it.

A client does not have to know the internal structure of the server. The only important thing is that client and server share the same language (or *protocol*). The protocol has to be established before any communication can take place. For the web, the protocol is the Hypertext Transfer Protocol (HTTP) [Mozilla 2019b].

In a web application, a web browser (client) might access a web server providing a service based on a database. After some user input, the web browser sends a request to the server, and then, depending on the type of request, the server accesses or queries the database in order to prepare a response and send it back to the web browser. In most cases, development of a web application consists of *frontend* and *backend* development. Frontend development covers the implementation of everything the end user will see in a web browser. Backend development covers the implementation of server-side logic. *Fullstack* development is a term encompassing both frontend and backend development.

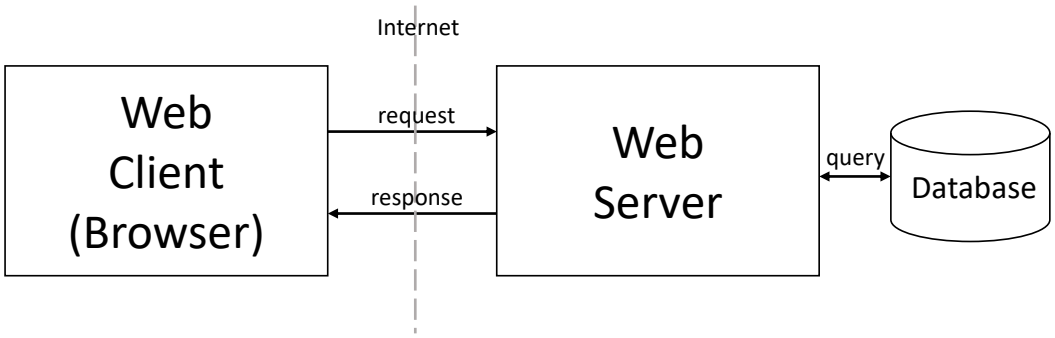


Figure 4.1: The client-server model on the web.

4.2 JavaScript

JavaScript (JS) is a programming language which is compiled during program execution [Marquis 2016]. It is a high-level programming language which has first-class functions (functions are treated as any other variables). It supports both the object-oriented and functional programming paradigms. It uses parts of the syntax of the C programming language (`if`, `while` and `switch` commands to name a few). JS was first introduced in 1995, as part of the Netscape Navigator browser, and was intended to program interactivity on websites. JavaScript has nothing to do with the programming language Java. The idea to name it JavaScript was done for marketing purposes [Haverbeke 2018]. Today, JS is integrated into most web browsers, but is also present on servers, and has been ported to all kinds of devices.

JS is the most popular programming language in frontend web development [Marquis 2016, Chapter 1], since almost all modern web browsers support JS by default. When a user requests a webpage containing JS code, the browser parses and executes the code. In the web browser, JS provides access to the browser's internal data structure, the Document Object Model (DOM) [Keith 2010, Chapter 1].

JS also has APIs to access device functionality such as location, local storage, drag-and-drop, camera, and microphone, although this access is often limited or requires explicit permission from the user. Without such restrictions, JS would be able to run other programs, access files on the client device and read private data, with significant privacy implications.

4.3 Node

JavaScript originated as a programming language used to make web sites interactive, to be run inside a web browser. This works perfectly when it comes to frontend development, because JS is efficient enough to manipulate and render a webpage's content to the user.

In contrast, Node is a standalone JS runtime environment with non-blocking I/O, which executes JS code outside of the browser website [Hughes-Croucher and Wilson 2012, Chapter 1]. Node was released in May 2009 and is written in C++ based on Google's V8 JS engine [Hughes-Croucher and Wilson 2012, Chapter 1]. Node has been ported to all kinds of devices and environments. It is also often used as a server's environment to process requests coming from the frontend. This way, both frontend and backend development can be written with a common language, JS, which makes it simpler and less time-consuming for developers. Frontend developers can easily serve in a dual role as backend developers too.

```
var express = require('express');
var app = express();

app.get('/users', function(req, res) {
  res.send({ users });
});

app.post('/user', function(req, res) {
  addUser();
});

var server = app.listen(8080, function() {
  var port = server.address().port;
  console.log('App is listening at port ' + port);
});
```

Listing 4.1: A simple Node application using Express.

One of the significant features of Node is its single-thread logic [Node 2019], which means that a new thread is not created with every request. In practice, this means that when Node has to perform a request like fetching values from a database, the thread is not blocked, and operations are resumed once the response comes back. This use of non-blocking I/O enables Node to process thousands of connections to the server on a single thread.

With this in mind, it is possible to optimise the frontend of a web application by reducing the number of requests going from client to server. It is often the case that a client sends a series of requests to the server, one after another. These requests are often related to and rely on previous requests. This creates an overload, and there is always a risk that something can go wrong with the connection between client and server. One way to improve this situation is to move parts of the client logic to the server. The client sends only one request, and it is up to the server to do the rest of the job and send a final response back. Significally fewer requests reduce the risk of connection problems between client and server, and significally increase performance.

4.4 Express

Express is a web framework for Node [Express 2019]. It provides a layer built on top of Node, which makes Node development more efficient. One of the main advantages of Express is its route handling. Express can be seen as middleware which processes requests, making the handling of HTTP requests (get, post, put, delete) simpler and possible to implement with less code. This is done with the help of a routing table, which contains information about different paths and HTTP request methods attached to them. Express is part of the MEAN (MongoDB, Express, Angular, Node) stack [Mean.js 2019]. It is used on the client-side of an application to help handle communication between client and server. Listing 4.1 shows an example of the usage of Express in Node.

4.5 TypeScript

One of the disadvantages of JavaScript is that it does not support static type checking. To overcome this, a superset of JS was developed by Microsoft, called *TypeScript* [Microsoft 2019; Fenton 2018]. It is

open-source, and was initially released in 2012. TypeScript contains all the features of JS, and also adds support for static type checking. This means that developers can see warnings and errors in the code at compile-time rather than run-time. This is especially useful for larger projects, where there are many relations between objects, and often multiple developers. If the project used plain JS, warning and errors would only be noticable at run time.

In TypeScript, it is also possible to write classes similar to those available in C++. A class can have public, private, and protected members, and a constructor. This supports the object-oriented programming paradigm, which is often better suited for larger projects.

TypeScript code is actually transpiled down to a particular version of JS code for execution. This is done as part of the build process, and then the resulting JS code is sent to the browser. This makes sense, because modern web browsers have a JS runtime engine built into their core. A disadvantage of using TypeScript is its initial learning curve. It is a programming language with a different syntax compared to JS, and it adds new object-oriented features.

Back in 2012 when TypeScript was initially released, there was only IDE support for Microsoft Visual Studio, which was only available on the Windows operating system. Today, TypeScript is supported in many IDEs on many different operating systems, such as OS X and Linux. Some of the most popular frontend web frameworks such as Angular are now written in TypeScript.

4.6 jQuery

jQuery [jQuery 2019] is a JavaScript framework which extends the core functionality of JS, and was initially released in August 2006. It simplifies programming with JS, since it reduces the amount of JS code necessary to implement various tasks such as DOM manipulation, event handling, animations, and Ajax [jQuery 2019]. To use jQuery, one simply has to include jQuery.js in HTML's script tag.

Since each web browser has the freedom to implement JS in its own way, a website which runs correctly in one browser, can have problems in a different browser. To overcome this problem, jQuery provided a solution for cross-browser issues such as DOM traversal and manipulation, selection of DOM elements, Ajax logic, and event handling. Therefore, choosing jQuery over plain JS reduced the chance of a website behaving differently in different browsers.

One of the disadvantages of using jQuery is the additional size of the project. jQuery version 3.4.1 adds around 84 kilobytes of code (minified), which will increase the initial website loading time. jQuery is still used by more than 73% of all websites [W3Techs 2019a], making it the most popular frontend framework. However, many of those are legacy sites with old technology. For a new web application, jQuery is now obsolete, because many of its features are now native in TypeScript or ES6 and ES7.

4.7 Bootstrap

Bootstrap is an open-source framework used for development with HTML, CSS, and JS [Bootstrap 2019; Jakobus 2018]. It is a set of tools with prebuilt components, which can be used for faster website development [Bootstrap 2019]. Initially, it was named Twitter Blueprint. The original idea was to combine common components and maintain consistency among internal Twitter projects. It was renamed Bootstrap, and officially released in August 2011. The Twitter team responsible for the first version of Bootstrap continued working on it together with a large team of developers across the world. Bootstrap's repository is one of the most popular on GitHub [GitHub 2019], and today more than 18 million websites use Bootstrap, which is more than 18% of all websites [W3Techs 2019b].

The current version of Bootstrap is Bootstrap 4, which comes with four files: bootstrap.css (152 kilobytes), bootstrap.js (56.7 kilobytes), popper.js (20.5 kilobytes), and jquery.js (68.2 kilobytes). To start

```
<link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.
    css">

<script
src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.4.0.min.js"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js
bootstrap.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/
    popper.min.js"></script>
```

Listing 4.2: HTML code for including files for Bootstrap 4.

using Bootstrap in a website, it is enough to include the CSS file. If the website makes use of Bootstrap's dynamic operations, it must include the JS files as well (bootstrap.js, popper.js, jquery.js). Listing 4.2 shows how to include Bootstrap in the HTML file.

Bootstrap 4 uses ems or rems for defining most sizes. However, pixels are still used for defining screen breakpoints. Bootstrap contains more than 100 classes which can be given to an HTML element. These classes are used to style buttons, position elements, make icons, create tables, and much more. However, one of the most popular Bootstrap's features is its responsive grid system. It is a way of organising a webpage's content into rows and columns. The idea is to divide the page into 12 columns, and then use one of the following classes to position the content: `col-xs-{num}`, `col-sm-{num}`, `col-md-{num}`, and `col-lg-{num}`, where `num` is a number between 1 and 12. Different classes represent different screen widths:

- xs as extra small (<576px)
- sm as small (>576px)
- md as medium (>768px)
- lg as large (>992px)
- xl as extra large (>1140px)

It is also possible to keep columns empty by using an offset. For example, `col-sm-offset-2` would keep the two columns empty. The code for a responsive grid in Bootstrap is shown in Listing 4.3 and the resulting UI grid in Figure 4.2.

Bootstrap is now becoming obsolete. It still depends on large parts of jQuery which is itself obsolete. Responsive grids can be built more easily and with less overhead using the new CSS Grid module [Mozilla 2019a]. Interactive UI components can be built with Vue or one of the other frontend frameworks.

```
<div class="container">
    <div class="row">
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
        <div class="col-md-4"></div>
        <div class="col-md-1"></div>
        <div class="col-md-1"></div>
        <div class="col-md-1"></div>
        <div class="col-md-6"></div>
        <div class="col-md-3"></div>
        <div class="col-md-offset-1 col-md-10"></div>
    </div>
</div>
```

Listing 4.3: HTML code for producing a Bootstrap 4 UI grid.

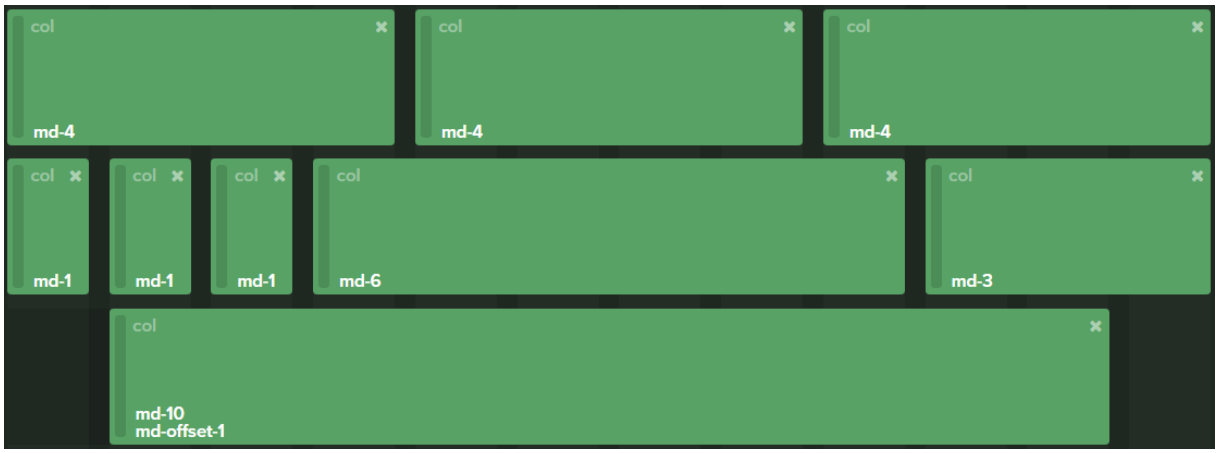


Figure 4.2: The Bootstrap UI grid produced by the code in Listing 4.3 [Screenshot of http://shoelace.io/, made by the author of this thesis].

4.8 MongoDB

MongoDB is a document-oriented database which stores information in JSON-like documents [MongoDB 2019b; Giamas 2019]. It is a non-relational (NoSQL) type of database, because it supports adding unstructured data and is not built upon a strict model schema [Chodorow 2013, Chapter 1]. MongoDB development started in 2007, and it was initially released in 2009. Currently, it is the most popular NoSQL database, and is used by companies including Google, Facebook, and PayPal [MongoDB 2019a].

A MongoDB document contains data consisting of a list of key-value pairs. A single document can be compared to a row in a relational database's table: document keys represent columns in the table, while values represent data in the row. A collection of these documents is similar of an SQL database table. The main difference between a collection of documents and an SQL database table is the fact that not all documents have to have the same set of keys. Consider for example, the following two documents:

```
{firstName: "John", lastName: "Doe", age: 27}

{firstName: "John", lastName: "Doe", city: "London"}
```

The model schema consists of first name, last name, age and city, but not all of these keys are used in both documents. Still, both documents are valid and will be stored in MongoDB. This would not be possible in an equivalent relational database such as SQL.

MongoDB's auto-scaling feature [Giamas 2019, Chapter 1] allows a situation when not all documents have to have the same data format. One can construct an initial data model, and expand it later with additional keys, without damaging previously saved data under an old data format. For example, if an additional key is added to the model, all previously inserted documents would remain unchanged. It is also possible to make a schema migration script, which could update the previously inserted documents, in order to make them compatible with the newest data model.

One disadvantage of using a MongoDB database instead of a relational database are the larger data sizes. This comes from the fact that each document contains information about the key under which a certain value is being saved. Querying is also not as flexible as in relational databases, for example, it is not possible to use JOIN functionality.

4.9 Single-Page Application Frameworks

Websites were initially built as a collection of HTML documents, where each document represented a part, or a page of the website. This meant that whenever a user wanted to see a different part of the website, an additional request was sent to the server in order to retrieve a new HTML document (or web page). As the complexity of websites increased over time, and they evolved into more sophisticated web applications, the number of requests which were being sent to the server also increased. This in turn increased the chance of problems occuring in the communication between client and server (waiting time, connection problems...). Single-page applications (SPA) were introduced to solve this problem. A *single-page application* is a website which contains only one page, whose content can be changed dynamically, without loading a new HTML document [Mikowski and Powell 2013, Chapter 1]. The URL in the browser's navigation bar can be changed manually in order to give the impression of subpages, but the website is never reloaded. Some of the most popular websites such as Facebook, Twitter, and Gmail are single-page applications. This does not come as a surprise, because these websites depend on a high level of user interaction, and views are constantly being updated.

Angular, React, and Vue are the three most popular JS frameworks for developing single-page applications. These frameworks use a routing system for navigation between the app's components. This means that each component has a state, which can be reflected in the browser's navigation bar.

4.9.1 Angular

Angular is a framework for developing web applications originating from Google [Google 2019a]. Initially, it was released as AngularJS in October 2010. In September 2016, an upgraded version under the name Angular was published, and this name was kept for all the upcoming versions. Angular is written in TypeScript and helps build single-page web applications (SPAs) [Freeman 2017]. The current version is Angular 8 [Google 2019a].

Developing an application with Angular requires usage of Angular-CLI. This is a tool for compiling the project, converting code to JavaScript, and finally combining the code into bundle files (`script.bundle.js`, `main.bundle.js`). These files contain compressed, minified, and optimised code, which is then included in the main HTML file. Angular-CLI can be used to compile the code in real time, propagating live changes to the browser for testing.

Developing a user interface in Angular consists of creating components. Each component resides in a folder containing one HTML, one CSS, and one TypeScript file. A component can be assigned to a selected route. Communication between components is done by injecting common services [Fain and

```
<div *ngIf="showMessage">Hello world</div>

export class HelloMessage {
    showMessage = true;
    constructor() {
    }
}
```

Listing 4.4: An example of Angular's *ngIf directive. The HTML element will be rendered if the variable showMessage has the value "true".

```
<div *ngFor="let item of items">{{item}}</div>

export class HelloMessage {
    items = ["Item 1", "Item 2", "Item 3"];
    constructor() {
    }
}
```

Listing 4.5: An example of Angular's *ngFor directive. *ngFor will iterate through a selected array and create as many DOM element as required.

Moiseev 2018, Chapter 5]. If one component manipulates data from the service, other components can notice the changes immediately. Angular also has a powerful collection of directives, which can be used in HTML elements [Google 2019b]. The most commonly used directives are:

- *ngIf for conditional rendering of a DOM element, as shown in in Listing 4.4
- *ngFor for iterating through an array of items, as shown in Listing 4.5

4.9.2 React

React is a JavaScript framework for building single-page applications [Facebook 2019]. The interactive UI is efficiently updated after any component's data changes. It was created by Facebook software developer Jordan Walke, and officially released in May 2013. Today, it is used by more than one million websites.

React uses a one-way data binding between model and view. This means that data in the view is updated automatically after model data changes. Another important feature of React is its use of a virtual DOM, used for tracking the changes. This allows it to more efficiently update the browser's actual DOM as necessary.

React's declarative programming style produces less code, and more readable code [Fedosejev 2015, Chapter 1]. This makes development in teams, or development on already existing projects more productive. A minimal React application is shown in Listing 4.6.

```
<div id="helloMessage"></div>

<script>
  class HelloMessage extends React.Component {
    render() {
      return <p>{this.props.message}</p>
    }
  }

  ReactDOM.render(<HelloMessage message="Hello!"/>,
    document.getElementById('helloMessage'));
</script>
```

Listing 4.6: Code for a minimal React application. A new React component is created. It contains a render method, which returns an HTML template. The ReactDOM is used to call the render method and update the view.

4.9.3 Vue

Vue is a lightweight JavaScript framework for building user interfaces [Vue 2019; Macrae 2018]. A minimal Vue application for the current version 2.6.0 consists of 33 kilobytes after being minified and gzipped. That is considerably smaller than comparable applications produced by other popular frontend frameworks (Angular's minimal application is six times larger). To start using Vue, it is enough to include it in a script tag:

```
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

Vue gained its popularity, because it can be added to an existing web application one small piece at a time. Every additional piece extends the app with some of Vue's functionality, but at the same time keeping the code clean and organised. It is also extremely straightforward to build a new application with Vue.

Vue implements both one-way and two-way data binding between model and view [Macrae 2018, Chapter 1]. This makes web application development more productive, since the developer does not have to update the model or view after the data changes: this is done automatically by the framework. Vue also supports component abstraction and nesting of components inside other components [Macrae 2018]. Listing 4.7 shows a simple Vue application.

4.9.4 Comparison

Table 4.1 shows a comparison of the AngularJS, Angular, React, and Vue frameworks. AngularJS is the most used framework and can be found in more than 1.1 million live websites [BuiltWith 2019]. That is more than the other frameworks combined. However, AngularJS is the oldest of the mentioned frameworks, and its successor Angular 2 is not as widespread. This could explain why React, which has almost three times less live websites, is more popular in Google searches, as shown in Figure 4.3.

```
<!--HTML-->
<div id="items">
  <ul>
    <li v-for="item in items">
      {{ item.text }}
    </li>
  </ul>
</div>

<!--JS-->
var items = new Vue({
  el: '#items',
  data: {
    items: [
      { text: 'Item 1' },
      { text: 'Item 2' },
      { text: 'Item 3' }
    ]
  },
})
```

Listing 4.7: Code for a minimal Vue application. A new Vue component is created, containing an array of items which are connected to the view's element with the id items.

	AngularJS	Angular	React	Vue
Release	2010	2016	2013	2014
Language	JavaScript	TypeScript	JavaScript	JavaScript
Learning	intermediate	challenging	intermediate	simple
Size of minimal app	172 kilobytes	200 kilobytes	100 kilobytes	20 kilobytes
Number of live websites	1,152,159	48,326	432,172	266,474

Table 4.1: Comparison of AngularJS, Angular, React, and Vue on 07 Sep 2019 [BuiltWith 2019].

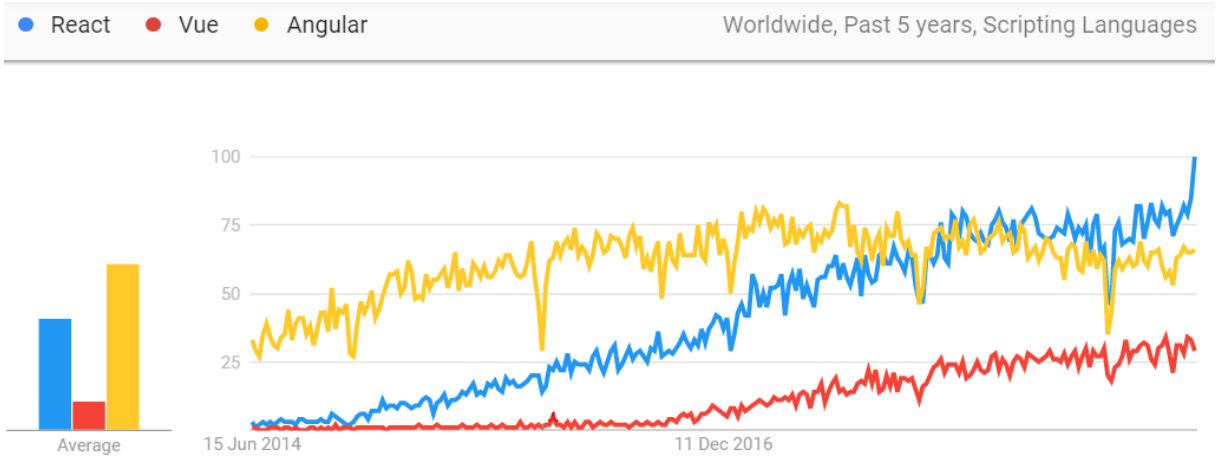


Figure 4.3: Visual representation of interest in the three most popular single-page application frameworks. "Numbers represent search interest relative to the highest point on the chart for the given region and time" [Google 2019c] [Screenshot of trends.google.com on 07 Sep 2019, made by the author of this thesis].

Chapter 5

TreeTest

TreeTest is an open-source web application for tree testing. It uses the following technologies:

- Angular 7 as a frontend framework.
- Node as a server.
- Express.js as a Node framework.
- MongoDB as a database.

These technologies are part of the MEAN stack (MongoDB, Express, Angular, Node), a fullstack JavaScript development solution [Mean.js 2019; IBM 2019]. The main strength of the MEAN stack is the "centralization of JavaScript as the main programming language" for both frontend and backend [Haviv 2016, Chapter 1]. In addition, the Bootstrap 3 framework is used in the frontend.

Angular is responsible for rendering the frontend part of the TreeTest app, handling the user's interactions, and sending requests to the Node server. The Node server waits for requests from the frontend, which are then serviced by a process which can include database data manipulation. All data is stored in MongoDB. The overall architecture of TreeTest is shown in Figure 5.1.

A typical workflow for the use of TreeTest might consist of the following steps:

1. The administrator of a TreeTest server creates and enables an account for a study owner.
2. The study owner creates a study and launches it.
3. The study link is shared with potential participants.
4. Participants open the study link and each perform a tree test.
5. The study owner views and analyses the study results.

A study owner requires a TreeTest account (username and password) to set up and run a tree testing study. Participants do not require a TreeTest account in order to take part in a study.

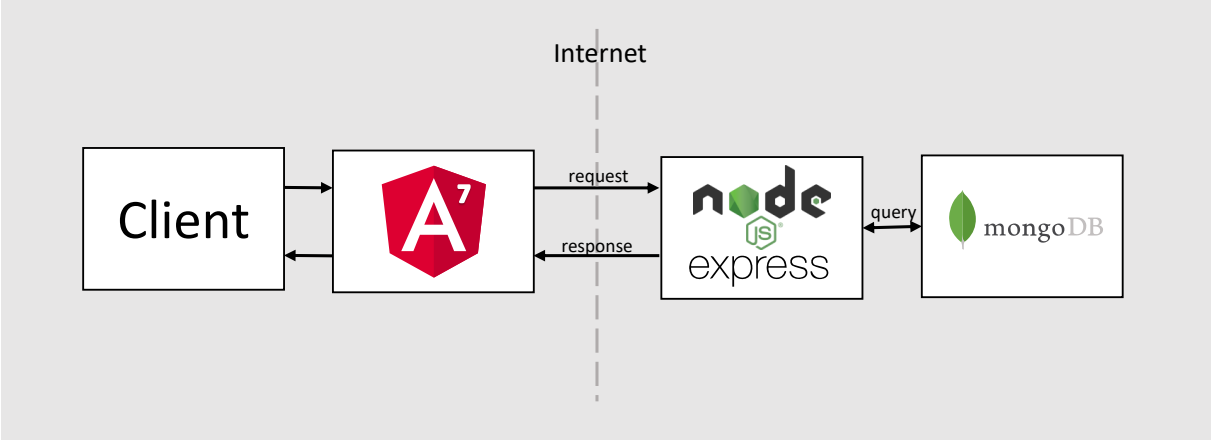


Figure 5.1: The overall architecture of TreeTest.

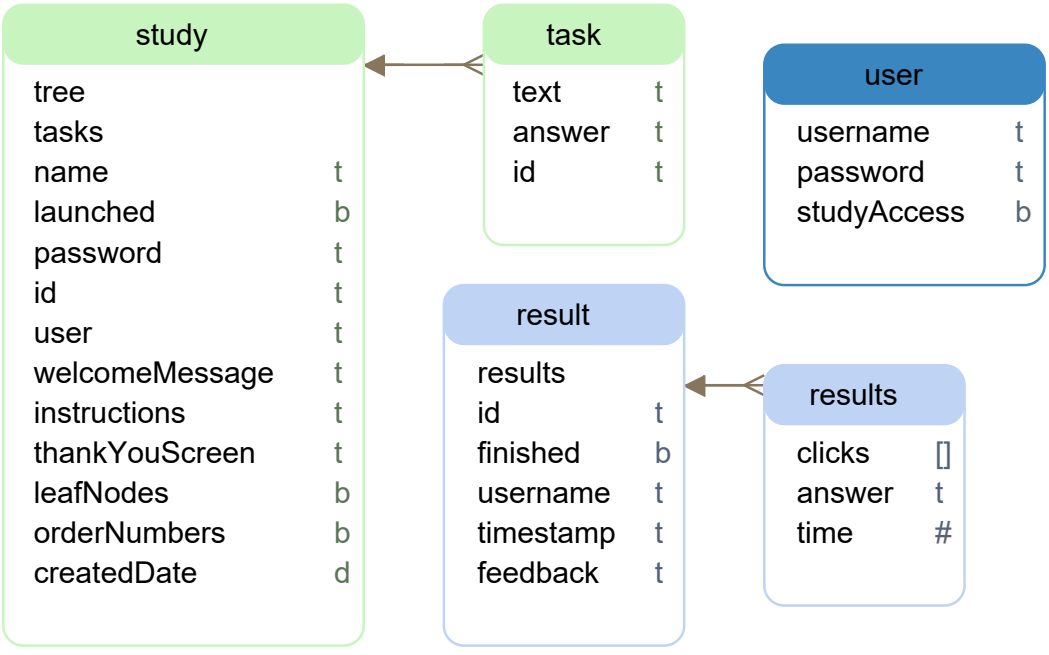


Figure 5.2: The MongoDB database schema used by TreeTest. `t` indicates a text string, `b` a boolean, `d` a date, `[]` an array, and `#` a real number. [Created using DbSchema [Wise Coders 2019] by the author of this thesis.]

5.1 TreeTest Database Schema

The MongoDB schema used by TreeTest is shown in Figure 5.2. It consists of the three models: `User`, `Study`, and `Result`. The MongoDB schemas for these models are shown in Listing 5.1, Listing 5.2, and Listing 5.3 respectively.

The attributes of the `user` model are:

- `username`. Unique identifier for each user.
- `password`. Used for logging in.
- `studyAccess`. It is false by default, and true after an admin gives enables study access for that user.

The attributes of the `study` model are:

- `name`. Study name.
- `password`. If it is set, it must be entered by a user who wants to do a study's test. Otherwise, anyone with a correct study URL can access it without any restriction.
- `launched`. If it is set to true, the study's test can be opened with a correct URL.
- `id`. Unique study identifier.
- `createdDate`. Study creation date.
- `tree`. An object containing the information hierarchy, with parent and child relations.
- `tasks`. An array containing tasks and correct answer for each.
- `user`. Study owner's TreeTest username.
- `welcomeMessage`. A welcome message shown to user before test starts.
- `instructions`. A message containing instructions for a test.
- `thankYouScreen`. A message shown to the user after the last task of the test is finished.
- `leafNodes`. If set to true, user can only select leaf nodes in information hierarchy as a task answer. Otherwise, non-leaf nodes can be selected as answers as well.
- `orderNumbers`. If set to true, user will see current task order number while performing a test. Otherwise, the current task order number will not be shown.

The attributes of the `result` model are:

- `studyId`. ID of the study to which result belong.
- `finished`. It is false if a user did not finish all the tasks in the test, and true otherwise.
- `username`. User identifier which is written by the user at the beginning of a test.
- `timestamp`. Time when a user finishes the test.
- `feedback`. Message which user can write at the end of the test.
- `result`. An object containing all the clicks in information hierarchy for each of the test tasks.

```
{
  username: { type: String },
  password: { type: String },
  studyAccess: { type: Boolean }
}
```

Listing 5.1: The MongoDB schema for a `User`.

```
{
  name: { type: String },
  password: { type: String },
  launched: { type: Boolean },
  id: { type: String },
  createdDate: { type: Date, default: Date.now },
  tree: { type: Array },
  tasks: { type: Array },
  user: { type: String },
  welcomeMessage: { type: String },
  instructions: { type: String },
  thankYouScreen: { type: String },
  leafNodes: { type: Boolean },
  orderNumbers: { type: Boolean }
}
```

Listing 5.2: The MongoDB schema for a `Study`.

```
{
  id: { type: String },
  finished: { type: Boolean },
  username: { type: String},
  timestamp: { type: String },
  feedback: { type: String },
  results: [
    clicks: [ node1, node2 ... ],
    answer: { type: String },
    time: { type: Float },
  ]
}
```

Listing 5.3: The MongoDB schema for the `Result` object, maintained for each participant who performs a tree test.

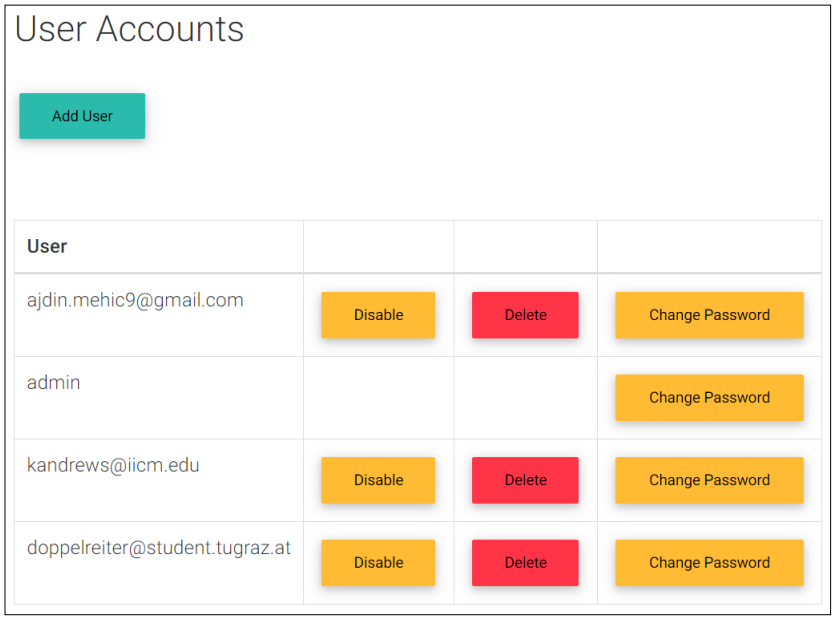


Figure 5.3: TreeTest. The Admin page. [Screenshot of TreeTest, made by the author of this thesis.]

5.2 Users and Roles

There are three different user roles in TreeTest:

- Admin. The Admin can add and delete users, change user passwords, and enable study access. There is only one predefined admin account.
- Study Owner. Creates and runs a study.
- Participant. Takes part in a tree testing study (does one test).

The Admin account is created during the initial application setup. The Admin can log in with the predefined credentials (username = admin, password = admin189m). The default password should be changed as soon as possible.

The Admin user has access to the Admin page, shown in Figure 5.3, which provides the following user management functions:

- Add a user. Done by clicking the Add User button. It is necessary to provide a username and initial password.
- Enable study access. Enables study access for a user, allowing the user to craete and run studies. It is done by clicking on Enable button for the corresponding user.
- Change password. Done by clicking the Change Password button for the corresponding user. A new password has to be provided.
- Delete a user. Done by clicking the Delete button for the corresponding user, and confirming the action in a popup dialog.

5.3 Creating a Study

A study is a complete tree testing project, containing an information hierarchy, navigation tasks, and basic information such as title, password, and instructions. A study is created through the New Study page, which has five tabs: Settings, Tree, Tasks, Messages, and Finish.

The Settings tab, shown in Figure 5.4, has the following properties:

- `Study Title`: the name of the study (required).
- `Study Password`: used by participants to access the study.
- `Selection of leaf nodes only`: enables or disables the possibility to only allow leaf nodes of the tree as an accepted answer.
- `Show task numbers to users`: shows or hides the task number while participants perform a study.
- `Study URL`: the link which participants can use to access the study.

The Tree tab, shown in Figure 5.5, is used to enter the information hierarchy (tree structure). The nodes (items) of the tree can be entered manually in the user interface, or can be imported from a CSV file. Semicolon and tab are also supported as delimiters.

After creating a tree, the study owner can proceed with adding tasks using the Tasks tab, shown in Figure 5.6. For each task, a question is formulated, and the location of its corresponding answer is defined in the tree. Once the first task has been created, the content of the tree can no longer be changed.

Messages and instructions are configured in the Messages tab, shown in Figure 5.7. The study owner can configure a welcome, instructions, and thank you message. All of these messages have some initial prefilled text, which can be kept or changed.

The Finish tab shown in Figure 5.8 displays a confirmation message that a new study has been created. A link to the Studies page is provided, which returns the study owner to their Studies page. During study creation, autosave is activated every time the study owner navigates between configuration tabs. This helps preserve data in situations where the study owner might close the app by mistake, before finishing the last step. Each study is persisted in the database in a Study object, as described in Section 5.1.

A study has to be launched in order to make it accessible to participants. The study owner can do this on the Studies page by clicking the Launch button. Once the study is launched, participants can open the study using the corresponding link.

The Studies page is used as the landing page when a study owner logs in. An overview of all studies belonging to the current user is shown in a table. Each row shows the Study Name, Study URL, and Edit, Launch/End, Delete, and Preview buttons, as shown in Figure 5.9.

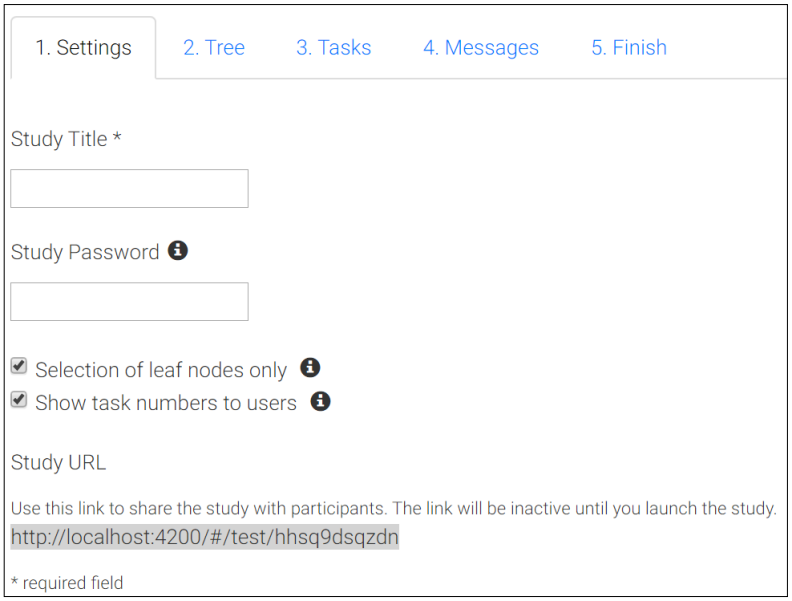


Figure 5.4: The Settings tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

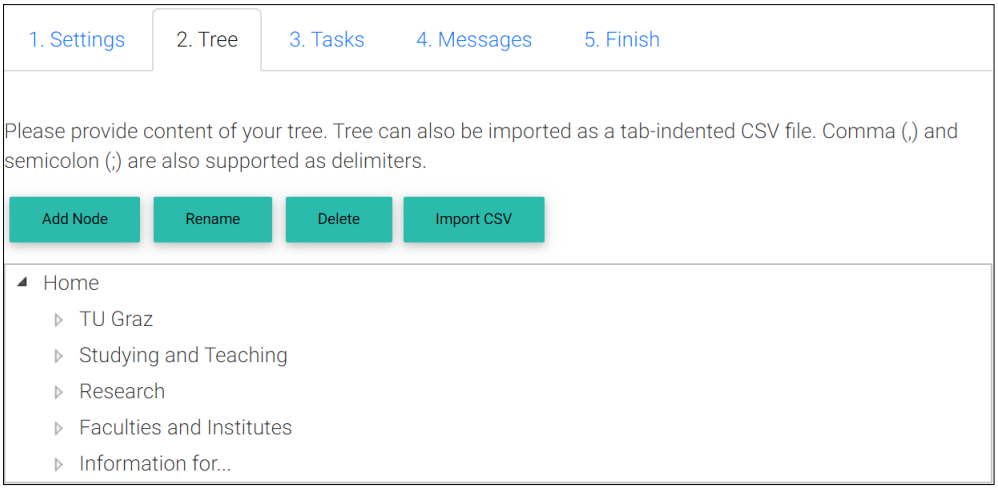


Figure 5.5: The Tree tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

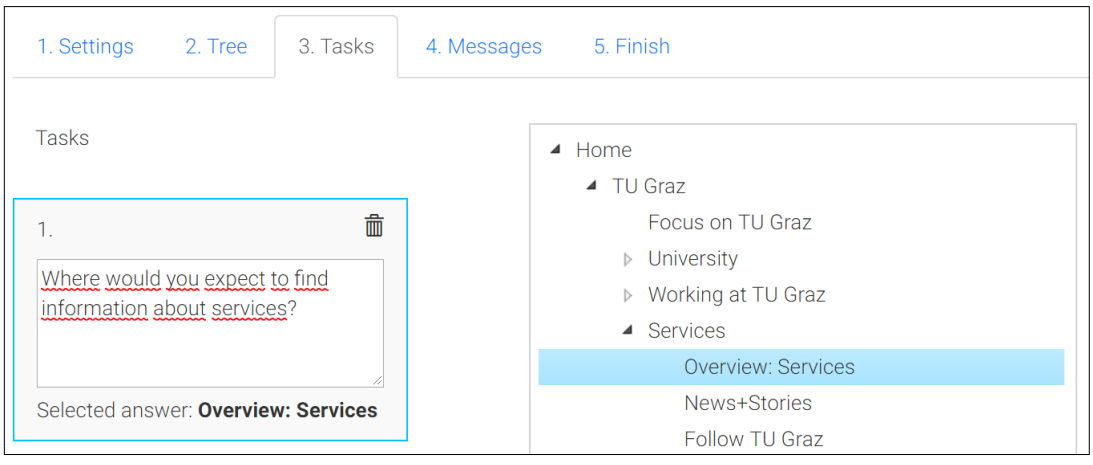


Figure 5.6: The Tasks tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

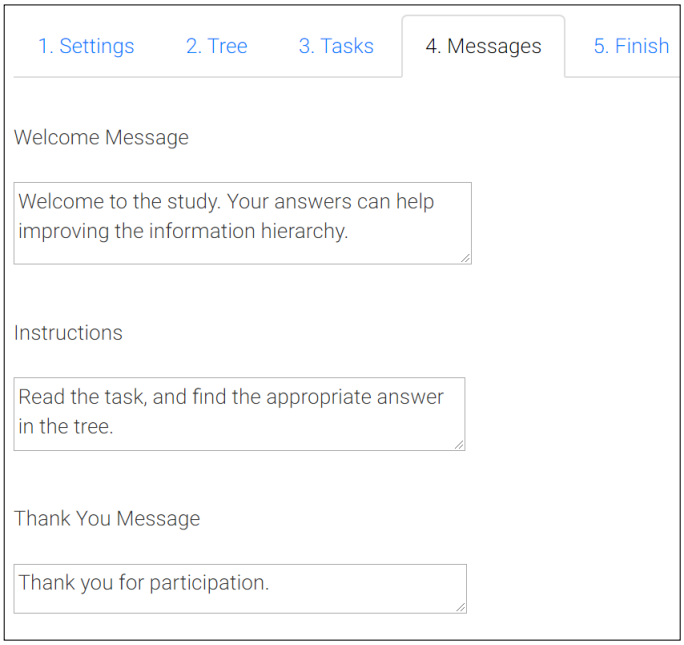


Figure 5.7: The Messages tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

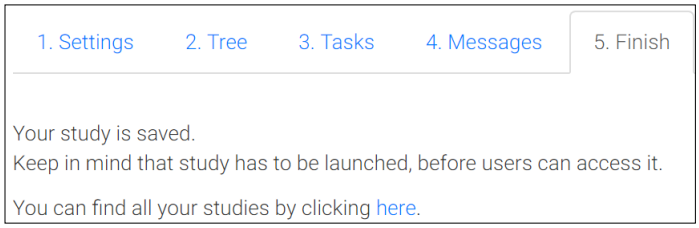


Figure 5.8: The Finish tab of the New Study page confirms that the new study was saved after the four configuration steps. [Screenshot of TreeTest, made by the author of this thesis.]

Figure 5.9: TreeTest. The Studies page is the landing page for study owners. [Screenshot of TreeTest, made by the author of this thesis.]

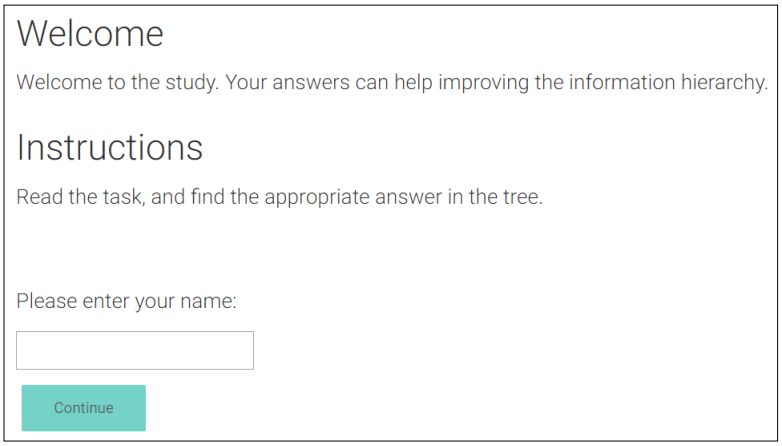


Figure 5.10: TreeTest. A participant first sees the Welcome screen. [Screenshot of TreeTest, made by the author of this thesis.]

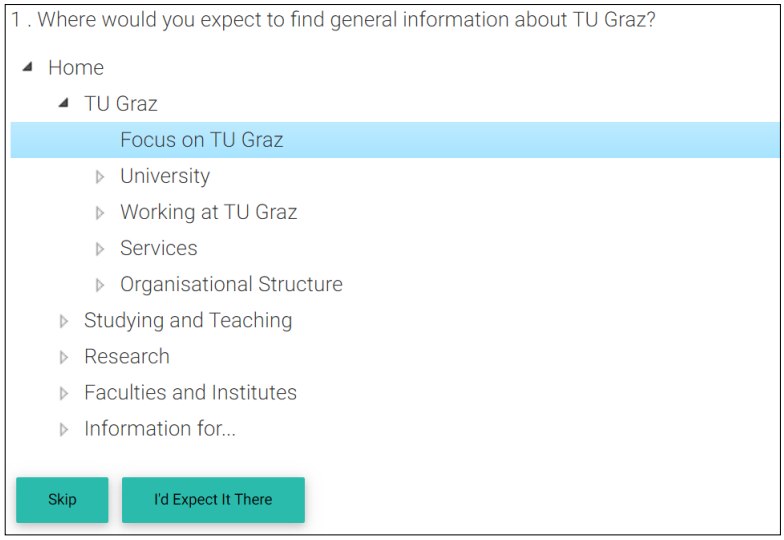


Figure 5.11: Participants are asked to indicate where in the tree they would expect to find a particular item. [Screenshot of TreeTest, made by the author of this thesis.]

5.4 Performing a Test

Participants are sent a study URL. After entering the URL in their web browser (and a security password where configured), they are presented with the Welcome screen shown in Figure 5.10. Participants are asked to enter their name. In the main part of the test, participants are presented with a series of tasks, in which they are asked to navigate through the information hierarchy and select a node in the tree where they would expect to find a particular item, as shown in Figure 5.11. At the end of the test, participants can optionally enter feedback for the study owner, and send it together with the results, as shown in Figure 5.12.

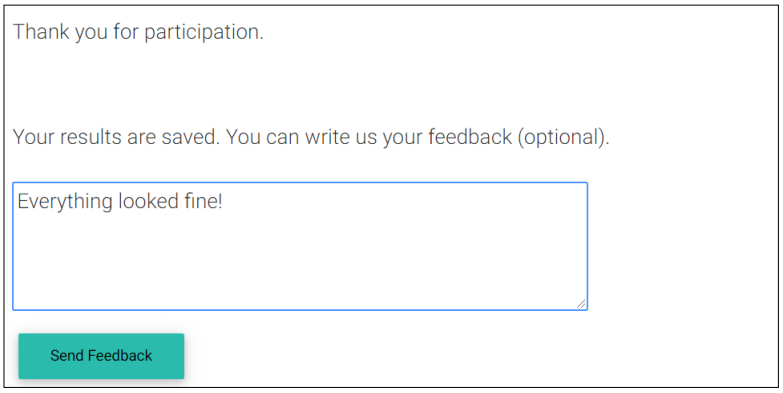


Figure 5.12: TreeTest. The Thank You screen of a test. [Screenshot of TreeTest, made by the author of this thesis.]

For each participant, the following data is collected during the test:

- Time taken to finish each task.
- Clicks in the information hierarchy.
- Clicks on the Skip button.
- Test abandonments.

The data is saved to the database for later use on the Study Results page. A `Result` object is maintained for each participant who performs a test.

5.5 Analysing the Results

After a sufficient number of participants have performed their tests, the study owner can view the study results on the Study Results page. The data is retrieved from MongoDB, by collating the results from all of the participants in a given study. The data is shown in various forms, including text, tables, charts, and graphs.

The Study Results page comprises the following tabs:

- Overview: Displays an overview of the results of a study, including the number of participants, average time needed to complete the test, and overall success and directness rate, as shown in Figure 5.13.
- Participants: Displays a Participants table, containing a summary of the results of each individual participant (date and time of test, duration of test, number of completed tasks, number of skipped tasks, number of correct tasks, and any feedback given), as shown in Figure 5.14. It is possible to exclude individual participants from consideration in the results using the Exclude button.
- Task Analysis: Shows the success rate, directness rate, and a path tree for each task, as shown in Figure 5.15. The path tree shows the paths taken by each user for a specific task. More frequently chosen paths are indicated by thicker branches. The correct path is shown in green. An example of a path tree is shown in Figure 5.16.
- Destinations: The Destinations table displays the number of clicks on each answer for each task, as shown in Figure 5.17.

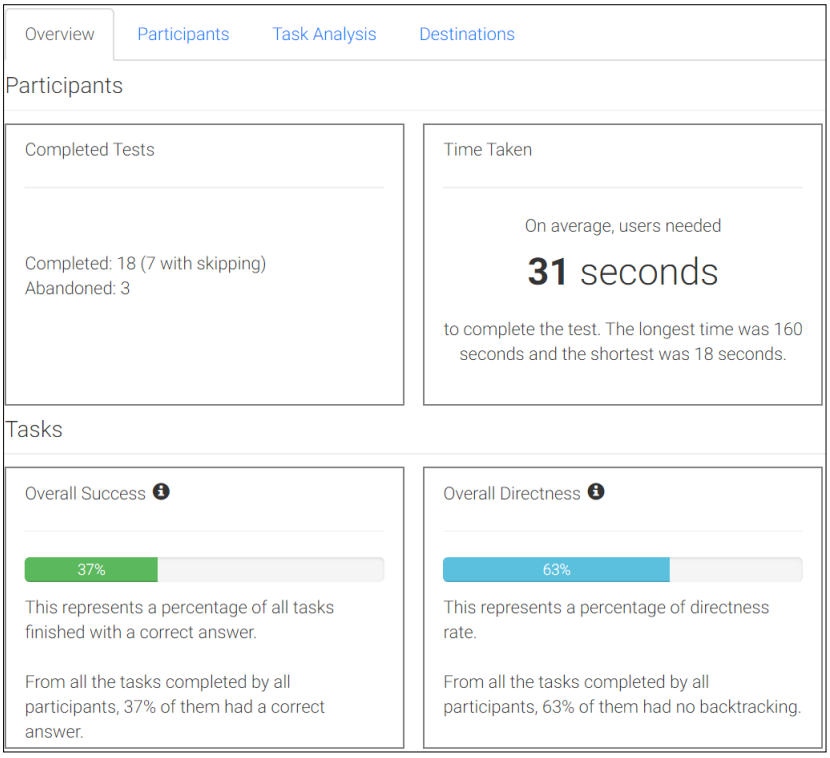
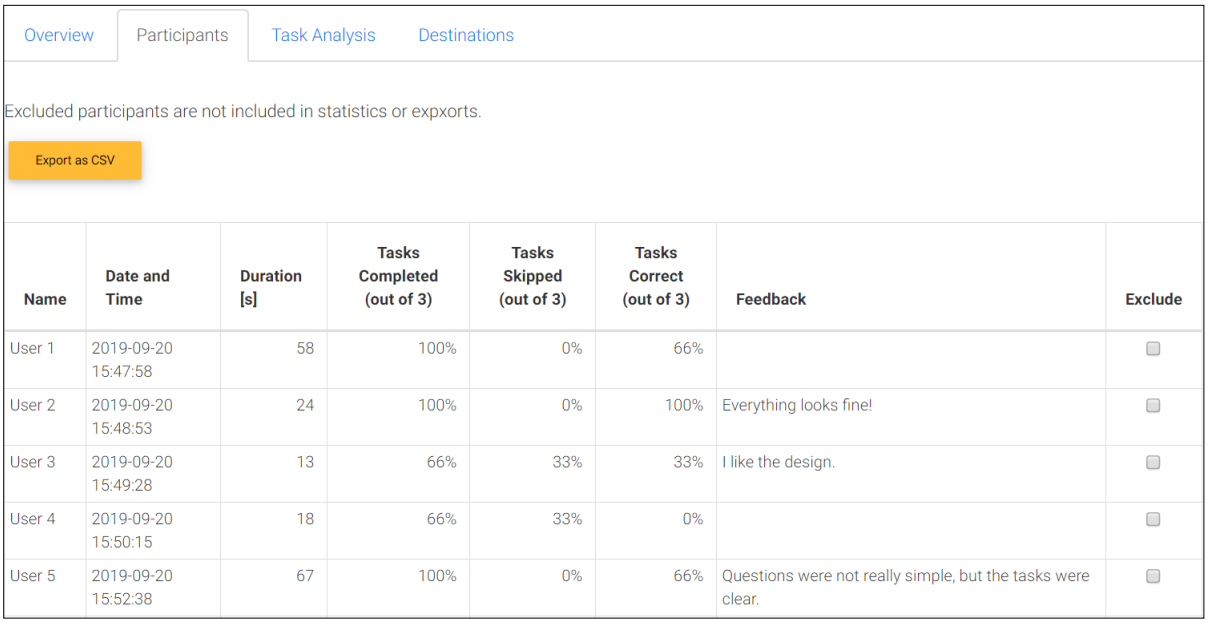


Figure 5.13: TreeTest. The Overview tab of the Study Results page. [Screenshot of TreeTest, made by the author of this thesis.]



Name	Date and Time	Duration [s]	Tasks Completed (out of 3)	Tasks Skipped (out of 3)	Tasks Correct (out of 3)	Feedback	Exclude
User 1	2019-09-20 15:47:58	58	100%	0%	66%		
User 2	2019-09-20 15:48:53	24	100%	0%	100%	Everything looks fine!	
User 3	2019-09-20 15:49:28	13	66%	33%	33%	I like the design.	
User 4	2019-09-20 15:50:15	18	66%	33%	0%		
User 5	2019-09-20 15:52:38	67	100%	0%	66%	Questions were not really simple, but the tasks were clear.	

Figure 5.14: The Participants tab of the Study Results page contains a summary of the results for each individual participant. The table can be exported as CSV. [Screenshot of TreeTest, made by the author of this thesis.]

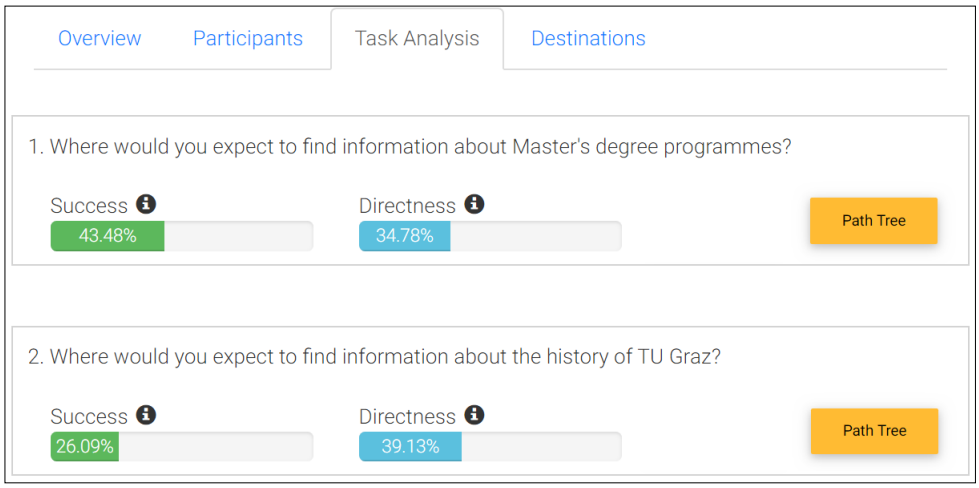


Figure 5.15: TreeTest. The Task Analysis tab of the Study Results page. Clicking the Path Tree button opens the path tree for that task. [Screenshot of TreeTest, made by the author of this thesis.]

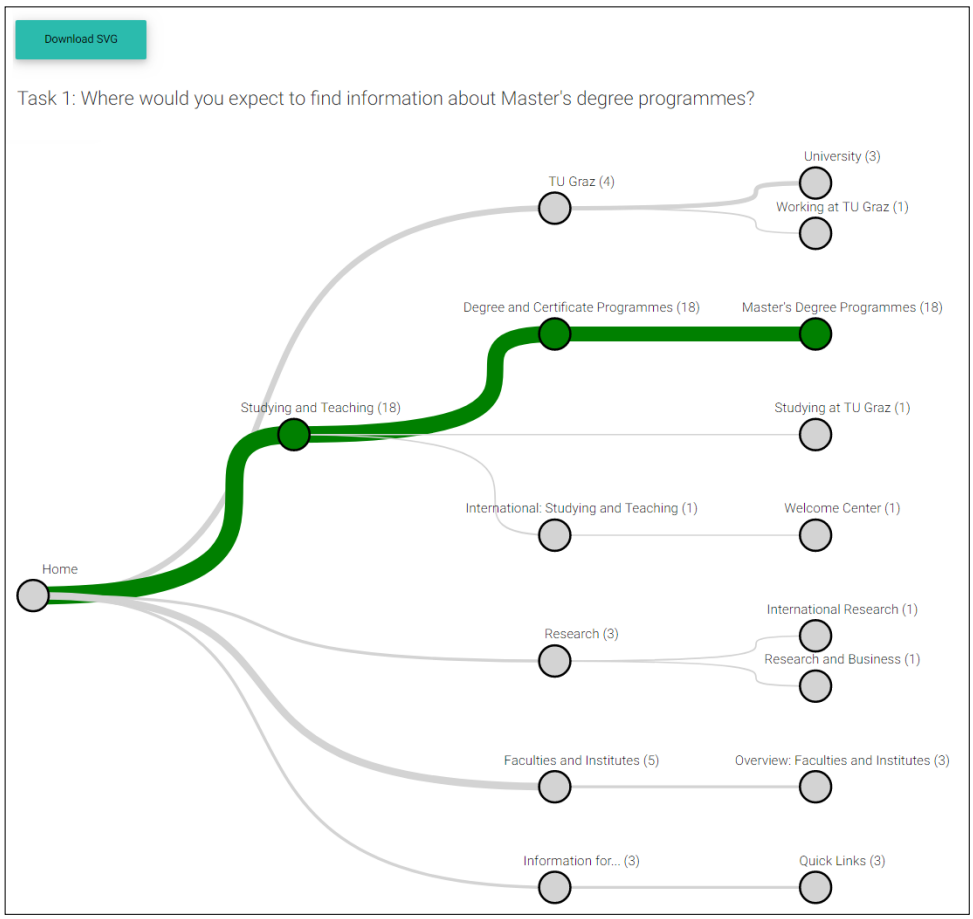


Figure 5.16: A path tree is used to visualise the paths the participants took while searching for the answer to a particular task. The correct path is shown in green. The thickness of each branch indicates how many users went down each path. The numbers in parentheses indicate the number of users who clicked on each node. The path tree can be exported as SVG. [Screenshot of TreeTest, made by the author of this thesis.]

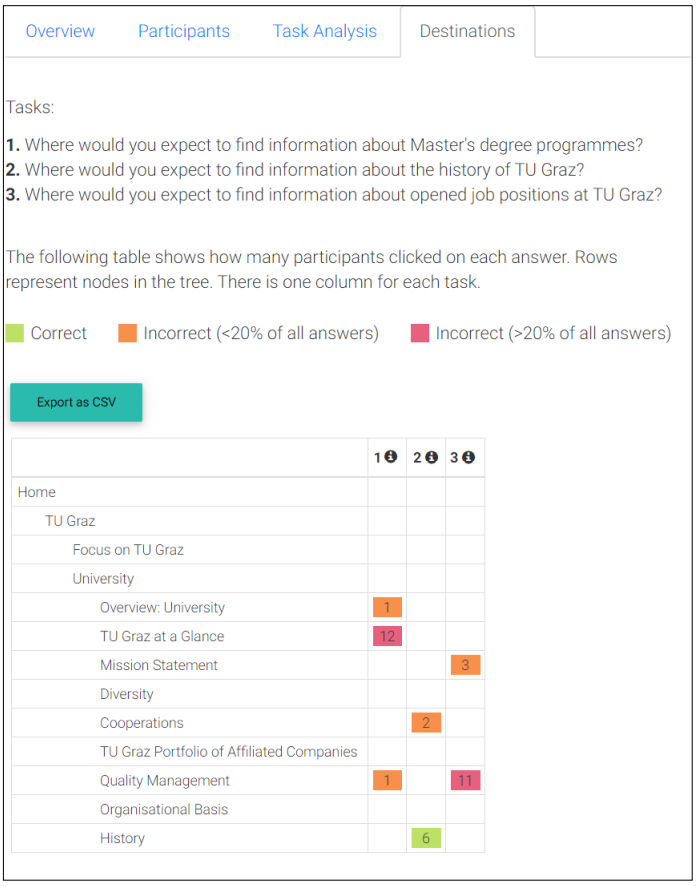


Figure 5.17: The Destinations tab of the Study Results page shows the number of clicks on each answer for each task. The table can be exported as CSV. [Screenshot of TreeTest, made by the author of this thesis.]

Chapter 6

User Testing

In order to gain an insight into TreeTest's user experience and potential usability issues, a thinking aloud test was performed, where test users were asked to verbalise their thoughts while performing typical tasks [Barnum 2010, Chapter 7]. The output of a thinking aloud test is a list of positive and negative findings. The negative findings indicate problems which need to be addressed.

Four users participated in the thinking aloud test, as shown in Table 6.2. Users worked on a Lenovo E590 laptop with 8 GB RAM running Windows 10. Test users used a keyboard and a mouse. Screen and voice were recorded using OBS Studio software [OBS Studio 2019]. The testing environment for TreeTest is shown in Figure 6.1.

A TreeTest study was created for the thinking aloud test. The information hierarchy of Graz University of Technology's external web site [TUG 2019a] was modelled in preparation for the thinking aloud test. Four tasks were defined, in which test users were asked to find appropriate answers in the tree, as shown in Table 6.1 (each user completed all tasks). In addition, two of the users were asked to take the role of a study owner and create a new study.

Before starting a thinking aloud test, each user was informed that there are no wrong answers, and that the test is meant to provide insight into the usability of the TreeTest interface. Figure 6.2 shows test user TP2 performing a test. Figure 6.3 shows test user TP4 creating a study.

After carefully analysing the recorded videos, a list of positive and negative findings was created, as shown in Tables 6.3 and 6.4. Positivity and severity ratings were assigned to each positive and negative finding, respectively. The findings are sorted in descending order of average (mean) positivity or severity. All of the findings were taken into consideration. The changes made to TreeTest included adding additional information for study configuration input fields, marking mandatory input fields, and updating message after creating a new study.

Figure 6.1: The testing environment for the TreeTest user study.

Task	Description
1.	Where would you find information about Master's degree programmes?
2.	Where would you find information about the history of TU Graz?
3.	Where would you find information about job vacancies?
4.	Where would you find information about rectorate of TU Graz?

Table 6.1: Overview of the tasks.

Test User	TP1	TP2	TP3	TP4
Date of Test	24 Aug 2019	01 Sep 2019	01 Sep 2019	19 Sep 2019
Time of Test	18:12	16:09	20:50	17:45
Alias	Eliot	Paula	Robert	Wolfgang
Sex	male	female	male	male
Age	31	45	39	26
Using PC since	16 years	17 years	11 years	10 years
Web Experience	14 years	15 years	11 years	10 years
Developer Experience	none	none	10 years	5 years

Table 6.2: Overview of the test users.

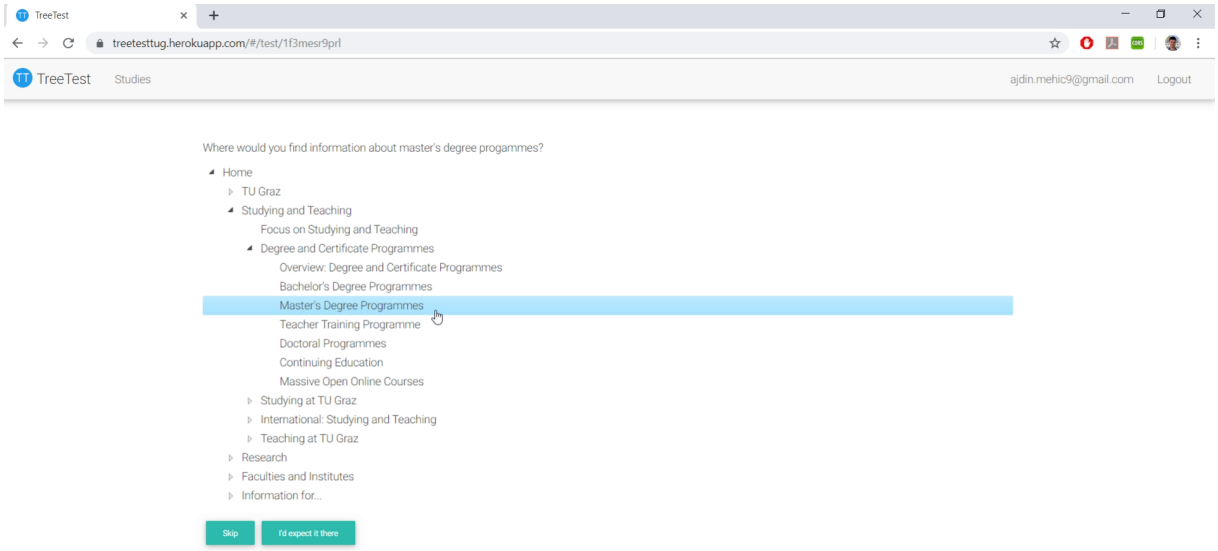


Figure 6.2: User TP2 performing a test

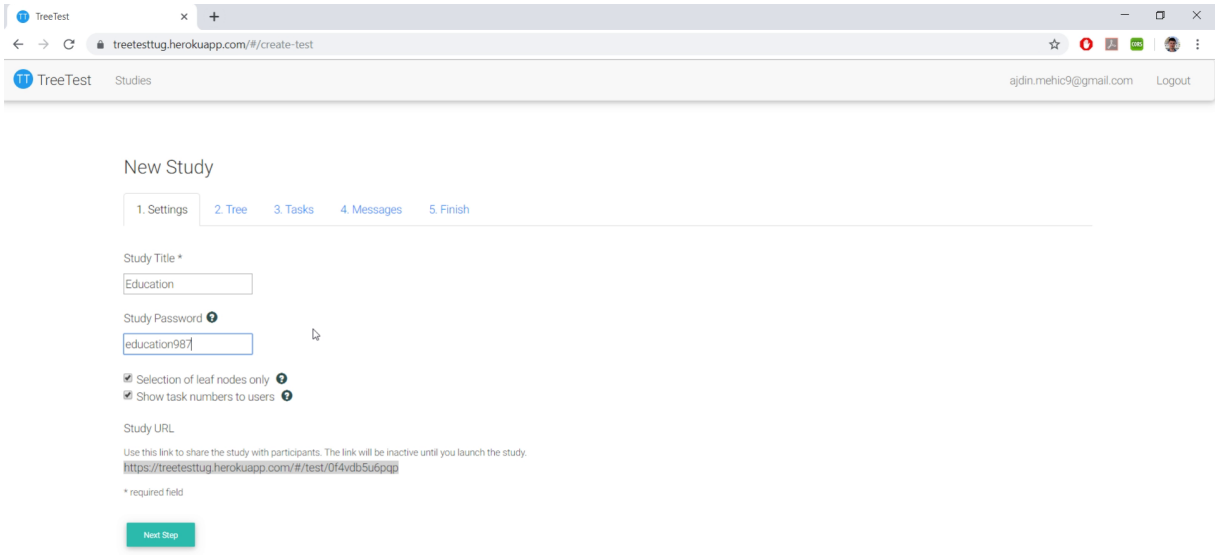


Figure 6.3: User TP4 creating a study

No.	Title	Description	Positivity
P1	Simple Design	Few details, and a consistent colours for buttons and text helps keep users focused on the content.	3.50
P2	Content is Well-Structured	Structure of pages is logical, and it is easy to find what is searched for.	3.50
P3	User-Friendly Test Process	Welcome message and introduction before the test begins helps users understand what will be necessary. Navigating through the tree while searching for an answer is flawless.	3.00
P4	Intuitive Steps in Creating a Study	Users know exactly how many steps are required in order to create a study, and it is clear what is the content of each step.	3.00

Table 6.3: Aggregated list of all positive findings, in descending order of average (mean) positivity.

No.	Title	Description	Severity
N1	Not Clear Which Input Fields are Mandatory	Users do not know if all input field are mandatory (for example, study password).	3.50
N2	Lack of Information about Configuration Parameters	It is not always clear what certain study configuration items are about (for example “Selection of leaf nodes only”, “Show task number to user”).	3.00
N3	Confusion on Last Study Creation Step	It is not always clear if the study was automatically created, or if there is an additional step that needs to be done after coming to the last step.	3.00
N4	Unclear if Results are Saved With No Feedback	Users have an opportunity to send feedback after doing a test, but it is not clear if test results are saved in a situation when user does not have any feedback to send.	2.00

Table 6.4: Aggregated list of all problems found, in descending order of average (mean) severity.

Chapter 7

Future Work

This version of TreeTest contains the basic functionality necessary for creating running and analysing a tree testing study, as described in Chapter 5. As a part of future work, additional features could be implemented:

- Request study permission. By default, a newly registered user (study owner) does not automatically have an enabled account. The admin has to explicitly grant permission for that user to create studies. A user should be able to send a request to obtaining such permission.
- Initial tutorial for new study owners. Once a study owner's account has been created and enabled, the new study owner should be able to see an interactive tutorial, explaining how the system works.
- How It Works page. This page should describe a simple TreeTest use case, containing study creation, doing a test, and analysing the results.
- User statistics. A logged-in study owner should be able to go to a Profile page, and see various study-related statistics including number of studies, study participants, tasks done, and total time taken.
- Migration to Angular 8. TreeTest is currently written in Angular 7. Angular 8 was released during the development of TreeTest and contains performance and application bundle improvements.
- Migration to Bootstrap 4. TreeTest currently uses Bootstrap 3. It should be upgraded to Bootstrap 4.
- Configurable study URL. It should be possible for a study owner to choose (at least the final part of) the study URL.
- Forgot Password page. A user should be able to request a new password.
- Change Password page. A User should be able to change their password.
- More sophisticated user account management. User should be able to manage their account in an Account Settings page.
- Inline path tree in Study Results page. In the Task Analysis tab of the Study Results page, the path tree for each task should be shown (in a smaller version) directly next to each task, rather than first having to be opened by clicking a button.
- Handling the browser's back button. In case the user clicks on the browser's back button, a warning message should be shown, indicating that unsaved data might be lost.

Chapter 8

Concluding Remarks

This thesis presented TreeTest, an open-source web application for testing information hierarchies (tree testing). Chapter 2 introduced the field of information architecture. Existing tools for tree testing were surveyed in Chapter 3. An overview of modern web technologies was then provided in Chapter 4.

TreeTest is built on top of the MEAN stack (MongoDB, Express, Angular, Node). The TreeTest implementation provides the basic functionality required for creating, running, and analysing the results of a tree testing study. A initial, formative user study (thinking aloud test) was conducted and the usability issues found were subsequently addressed.

In future work, the current TreeTest implementation can be built upon and extended in various ways to potentially provide an open-source alternative to existing commercial tree testing applications.

Appendix A

Administrator Guide

The TreeTest source code is stored in a Git (Gitlab) repository. If access to the repository is available, the following set of commands will clone the project, and install the required dependencies:

```
$ git clone git@git.isds.tugraz.at:kandrews/treetest.git
$ cd treetest/software/treetest
$ npm install
```

Assuming Node and MongoDB are installed on the local machine, starting the project locally is done through the following three steps:

1. Start and connect to MongoDB.
2. Start the Node server by running the command `node server.js`.
3. Start the Angular frontend by running the command `ng serve`.

The project can then be opened at `http://localhost:4200/`.

In order to deploy TreeTest on a dedicated server, Node and MongoDB first have to be installed on the server. Deploying TreeTest on the server is then done through the following steps:

1. In the root directory of TreeTest, run `ng build --prod` to create a frontend bundle in the `dist` directory. The contents of the `dist` directory should be deployed on the server, and the web application's URL should point to the `index.html` file contained in the `dist` directory.
2. The directory `server` and the file `server.js` should be deployed to the server. Then the Node server should be started with the command `node server.js`.

TreeTest can also be deployed using a Node hosting provider, such as Heroku [Salesforce 2019], which is particularly well-suited for hosting MEAN web applications.

For bug-fixing or further development, the TreeTest source code resides in two locations: the `src` directory for Angular frontend code, and the `server` directory for Node backend code. Changing Angular code will automatically trigger a rebuild, and the changes can be viewed live in the browser. After changing Node code, it is required to re-run the `node server.js` command, in order to incorporate the changes.

Appendix B

Study Owner Guide

A typical workflow for the use of TreeTest might consist of the following steps:

1. The administrator of a TreeTest server creates and enables an account for a study owner.
2. The study owner creates a study and launches it.
3. The study link is shared with potential participants.
4. Participants open the study link and each perform a tree test.
5. The study owner views and analyses the study results.

A study owner requires a TreeTest account (username and password) to set up and run a tree testing study. Participants do not require a TreeTest account in order to take part in a study.

B.1 Creating a Study

A study is a complete tree testing project, containing an information hierarchy, navigation tasks, and basic information such as title, password, and instructions. A study is created through the New Study page, which has five tabs: Settings, Tree, Tasks, Messages, and Finish.

The Settings tab, shown in Figure 5.4, has the following properties:

- `Study Title`: the name of the study (required).
- `Study Password`: used by participants to access the study.
- `Selection of leaf nodes only`: enables or disables the possibility to only allow leaf nodes of the tree as an accepted answer.
- `Show task numbers to users`: shows or hides the task number while participants perform a study.
- `Study URL`: the link which participants can use to access the study.

The Tree tab, shown in Figure 5.5, is used to enter the information hierarchy (tree structure). The nodes (items) of the tree can be entered manually in the user interface, or can be imported from a CSV file. Semicolon and tab are also supported as delimiters.

After creating a tree, the study owner can proceed with adding tasks using the Tasks tab, shown in Figure B.3. For each task, a question is formulated, and the location of its corresponding answer is defined in the tree. Once the first task has been created, the content of the tree can no longer be changed.

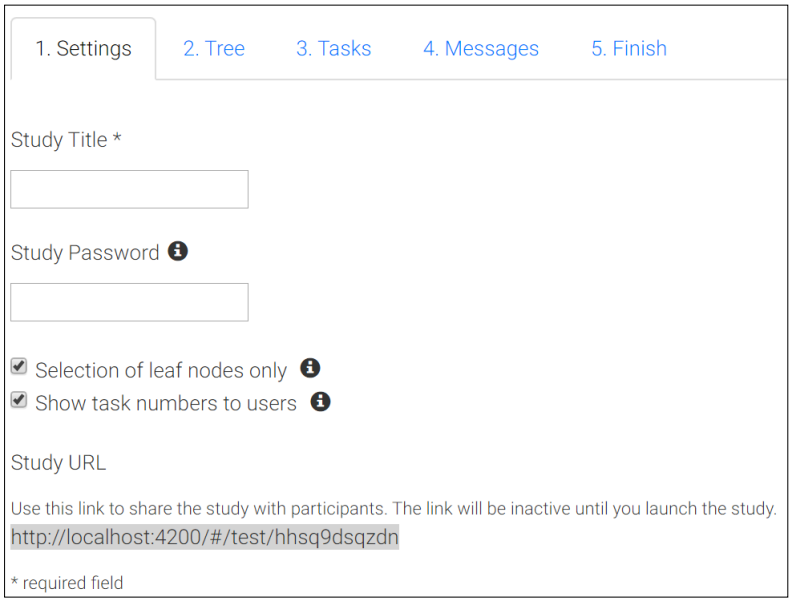


Figure B.1: The Settings tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

Messages and instructions are configured in the Mesages tab, shown in Figure B.4. The study owner can configure a welcome, instructions, and thank you message. All of these messages have some initial prefilled text, which can be kept or changed.

The Finish tab shown in Figure B.5 displays a confirmation message that a new study has been created. A link to the Studies page is provided, which returns the study owner to their Studies page. During study creation, autosave is activated every time the study owner navigates between configuration tabs. This helps preserve data in situations where the study owner might close the app by mistake, before finishing the last step.

A study has to be launched in order to make it accessible to participants. The study owner can do this on the Studies page by clicking the Launch button. Once the study is launched, participants can open the study using the corresponding link.

The Studies page is used as the landing page when a study owner logs in. An overview of all studies belonging to the current user is shown in a table. Each row shows the Study Name, Study URL, and Edit, Launch/End, Delete, and Preview buttons, as shown in Figure B.6.

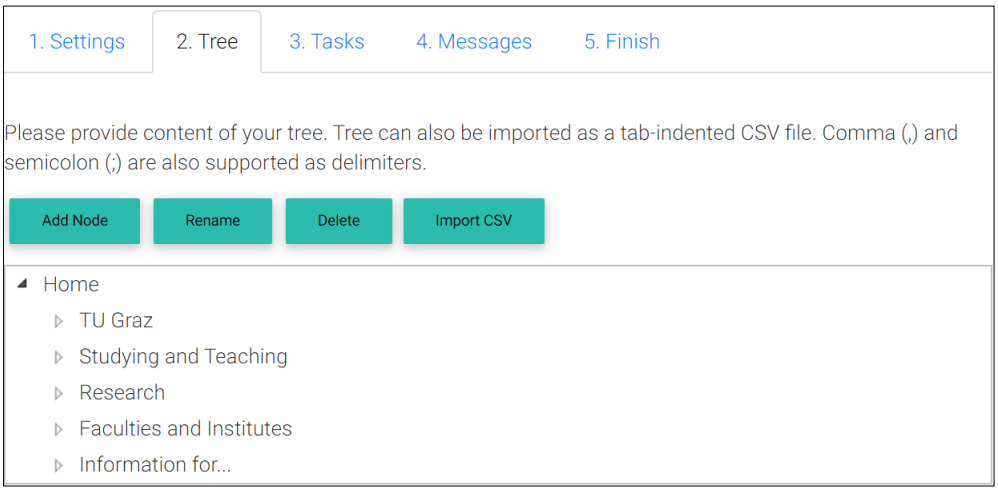


Figure B.2: The Tree tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

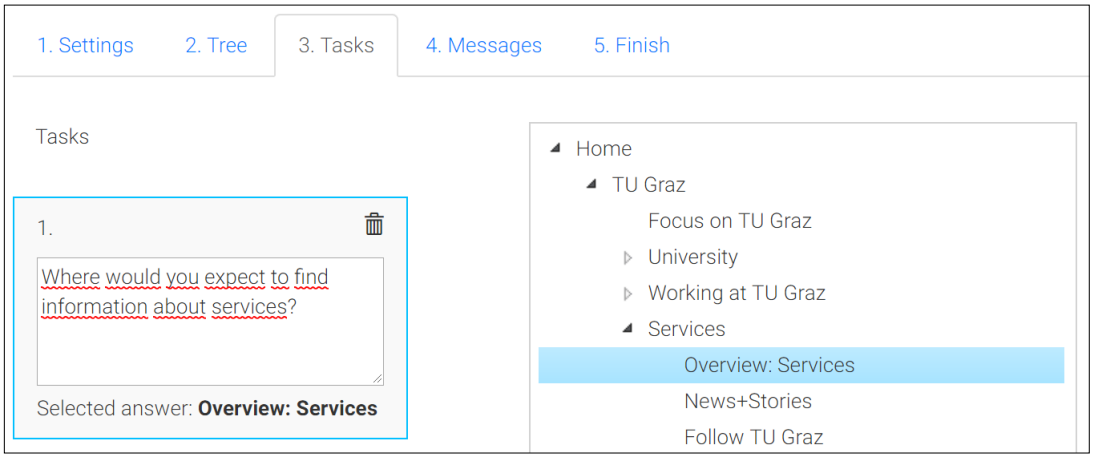


Figure B.3: The Tasks tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

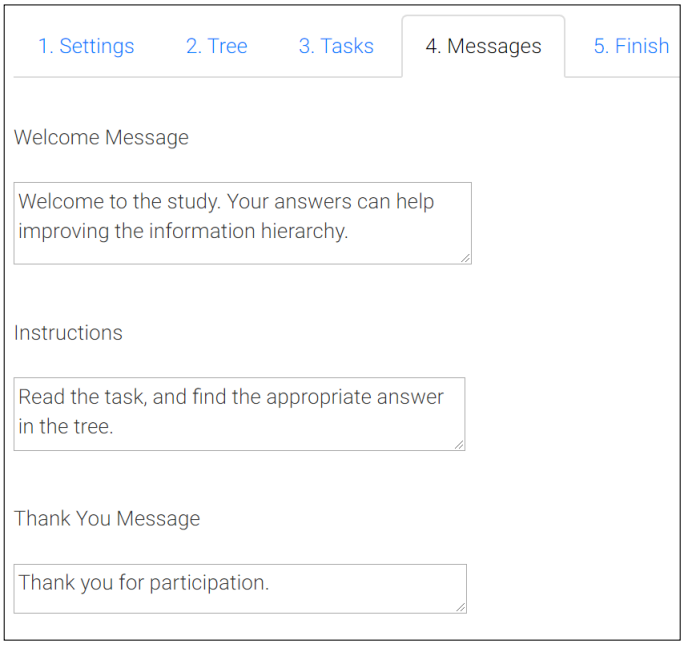


Figure B.4: The Messages tab of the New Study page during the process of creating a new study in TreeTest. [Screenshot of TreeTest, made by the author of this thesis.]

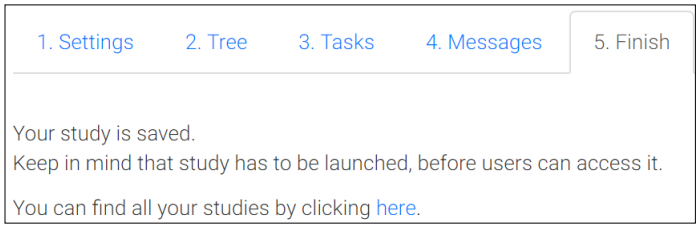


Figure B.5: The Finish tab of the New Study page confirms that the new study was saved after the four configuration steps. [Screenshot of TreeTest, made by the author of this thesis.]

Figure B.6: TreeTest. The Studies page is the landing page for study owners. [Screenshot of TreeTest, made by the author of this thesis.]

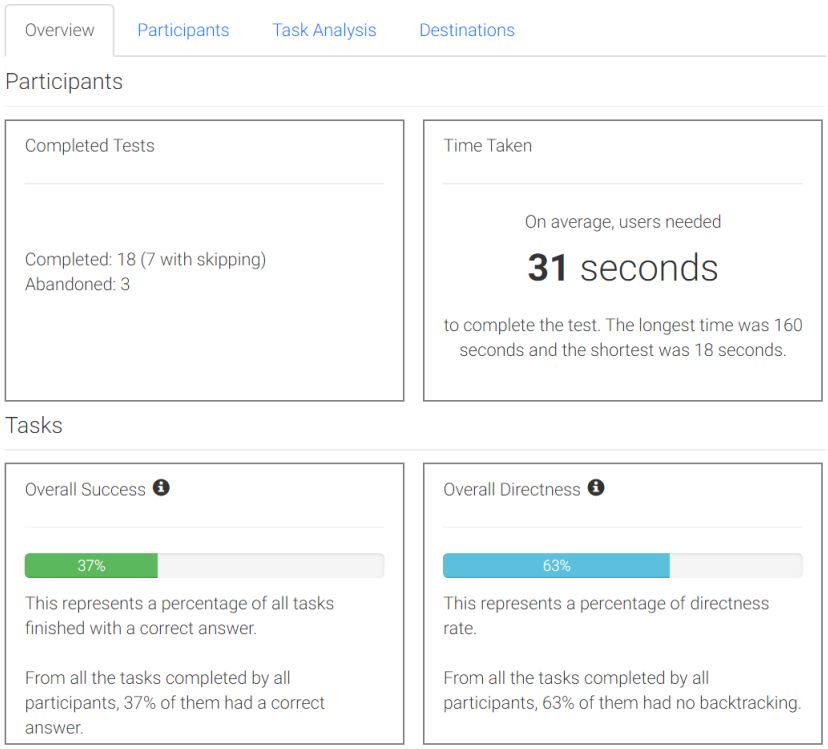


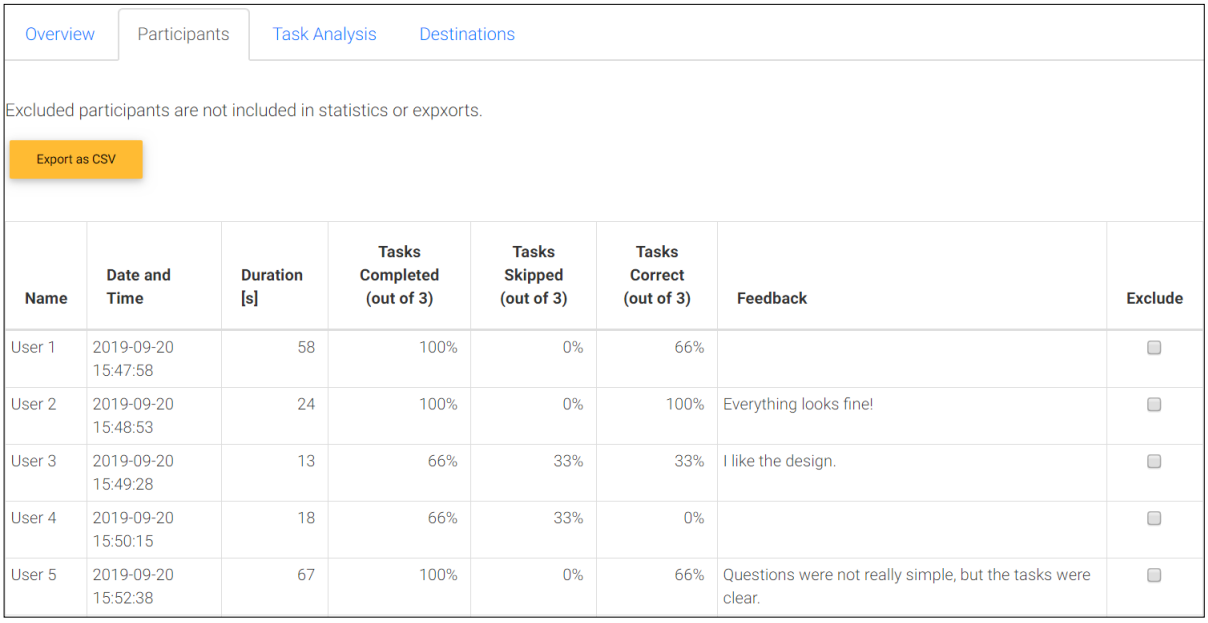
Figure B.7: TreeTest. The Overview tab of the Study Results page. [Screenshot of TreeTest, made by the author of this thesis].

B.2 Analysing the Results

After a sufficient number of participants have performed their tests, the study owner can view the study results on the Study Results page. The data is retrieved from MongoDB, by collating the results from all of the participants in a given study. The data is shown in various forms, including text, tables, charts, and graphs.

The Study Results page comprises the following tabs:

- Overview: Displays an overview of the results of a study, including the number of participants, average time needed to complete the test, and overall success and directness rate, as shown in Figure B.7.
- Participants: Displays a Participants table, containing a summary of the results of each individual participant (date and time of test, duration of test, number of completed tasks, number of skipped tasks, number of correct tasks, and any feedback given), as shown in Figure B.8. It is possible to exclude individual participants from consideration in the results using the Exclude button.
- Task Analysis: Shows the success rate, directness rate, and a path tree for each task, as shown in Figure B.9. The path tree shows the paths taken by each user for a specific task. More frequently chosen paths are indicated by thicker branches. The correct path is shown in green. An example of a path tree is shown in Figure B.10.
- Destinations: The Destinations table displays the number of clicks on each answer for each task, as shown in Figure B.11.



Name	Date and Time	Duration [s]	Tasks Completed (out of 3)	Tasks Skipped (out of 3)	Tasks Correct (out of 3)	Feedback	Exclude
User 1	2019-09-20 15:47:58	58	100%	0%	66%		☐
User 2	2019-09-20 15:48:53	24	100%	0%	100%	Everything looks fine!	☐
User 3	2019-09-20 15:49:28	13	66%	33%	33%	I like the design.	☐
User 4	2019-09-20 15:50:15	18	66%	33%	0%		☐
User 5	2019-09-20 15:52:38	67	100%	0%	66%	Questions were not really simple, but the tasks were clear.	☐

Figure B.8: The Participants tab of the Study Results page contains a summary of the results for each individual participant. The table can be exported as CSV. [Screenshot of TreeTest, made by the author of this thesis].

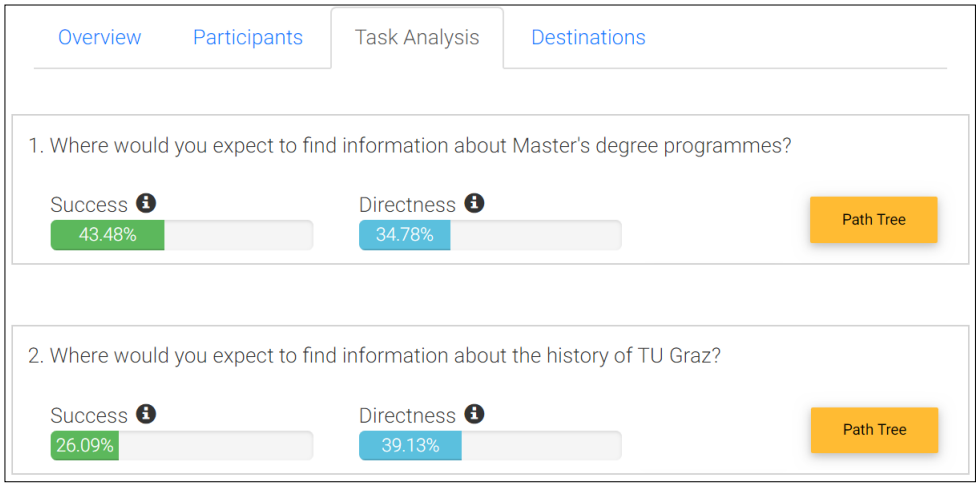


Figure B.9: TreeTest. The Task Analysis tab of the Study Results page. Clicking the Path Tree button opens the path tree for that task. [Screenshot of TreeTest, made by the author of this thesis].

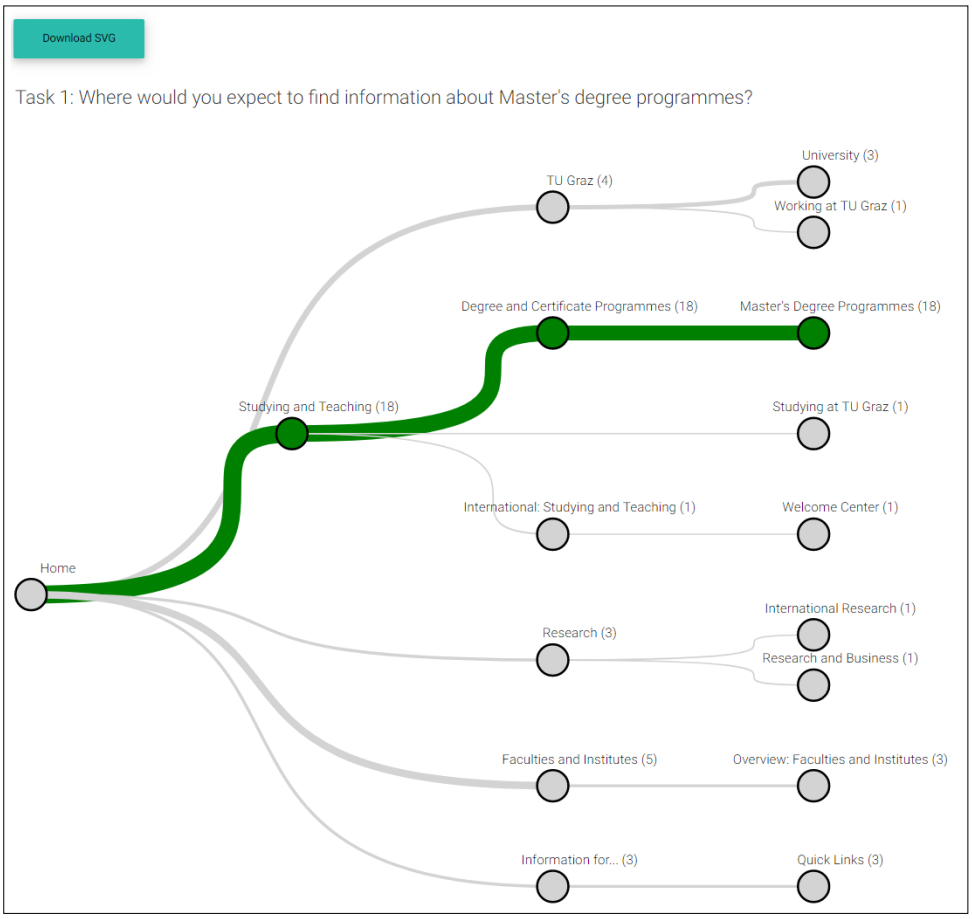


Figure B.10: A path tree is used to visualise the paths the participants took while searching for the answer to a particular task. The correct path is shown in green. The thickness of each branch indicates how many users went down each path. The numbers in parentheses indicate the number of users who clicked on each node. The path tree can be exported as SVG. [Screenshot of TreeTest, made by the author of this thesis].

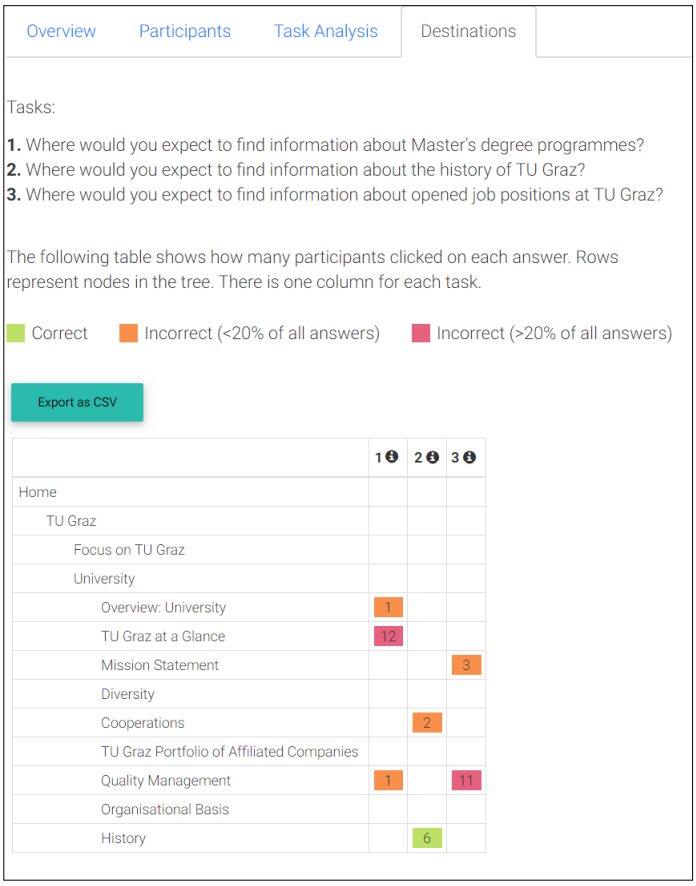


Figure B.11: The Destinations tab of the Study Results page shows the number of clicks on each answer for each task. The table can be exported as CSV. [Screenshot of TreeTest, made by the author of this thesis].

Appendix C

Participant Guide

If the study owner has set a security password, participants will be required to enter it before performing their test. This is done in order to make sure that only selected participants will have access to the study.

On starting a test, participants are asked to enter their name in the Welcome page, as shown in Figure C.1. At the end of the test, participants can optionally enter feedback for the study owner, as shown in Figure C.3. The main part of the test are the tasks in which the participant is asked to navigate through the information hierarchy, and propose an answer for each task, as shown in Figure C.2.

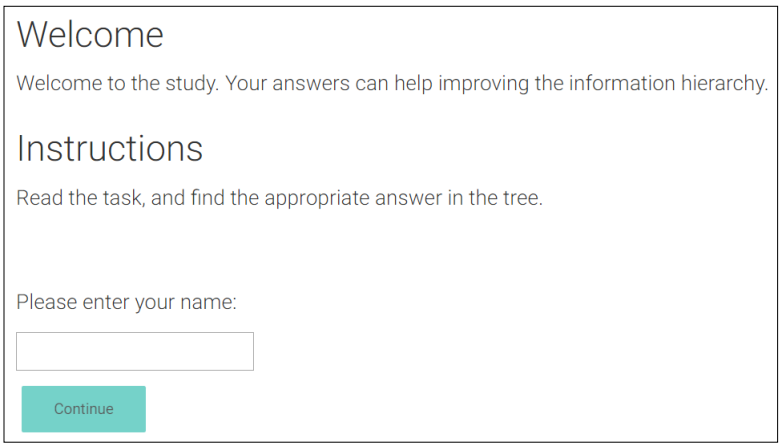


Figure C.1: TreeTest. The Welcome screen of a test is the first thing a participant sees. [Screenshot of TreeTest, made by the author of this thesis.]

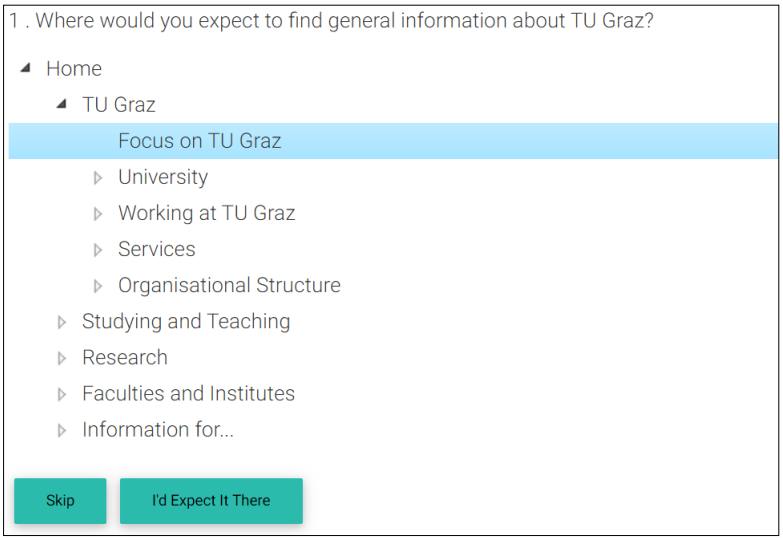


Figure C.2: Participants are asked to indicate where in the tree they would expect to find a particular item. [Screenshot of TreeTest, made by the author of this thesis.]

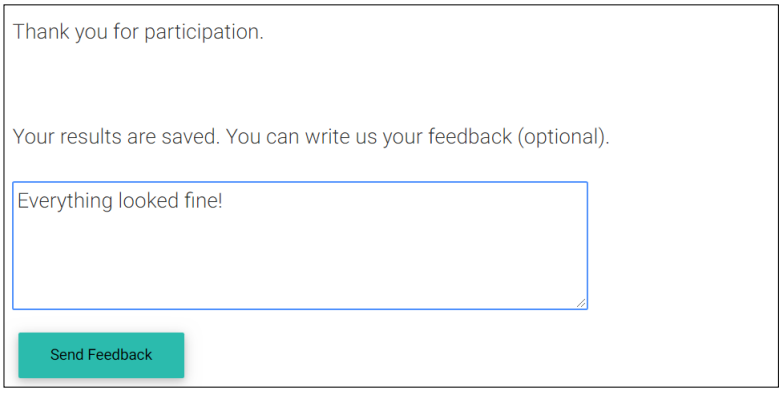


Figure C.3: TreeTest. The Thank You screen of a test. [Screenshot of TreeTest, made by the author of this thesis.]

Bibliography

Andrews, Keith [2019]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 24 Jan 2019. https://ftp.isds.tugraz.at/pub/keith/thesis/ (cited on page xi).

Barnum, Carol M. [2010]. *Usability Testing Essentials*. Morgan Kaufmann, 05 Nov 2010. 408 pages. ISBN 012375092X (cited on page 49).

Bootstrap [2019]. *Bootstrap*. 07 Sep 2019. https://getbootstrap.com/ (cited on page 28).

Borreguero Llorente, Pablo, Anna Lickl, Michael Planitzer, and Daniel Sudy [2017]. *Tree Testing in Information Hierarchies*. Survey Paper, IAweb, Winter Semester 2017. Graz University of Technology, 04 Dec 2017. https://courses.isds.tugraz.at/iaweb/surveys/ws2017/iaweb-ws2017-g2-survey-tree-testing.pdf (cited on pages xi, 7–8, 16).

Borreguero Llorente, Pablo, Anna Lickl, Michael Planitzer, and Daniel Sudy [2018]. *Tree Testing the tugraz.at/en Information Hierarchy*. Project Report, IAweb, Winter Semester 2017. Graz University of Technology, 04 Jan 2018. https://courses.isds.tugraz.at/iaweb/projects/ws2017/iaweb-ws2017-g2-project-tree-testing.pdf (cited on pages xi, 7–8).

BuiltWith [2019]. *Usage Statistics*. 07 Sep 2019. https://trends.builtwith.com/ (cited on pages 33–34).

Cederholm, Dan [2014]. *CSS3 for Web Designers*. 2nd Edition. A Book Apart, 2014. 142 pages. ISBN 1937557200 (cited on page 25).

Chodorow, Kristina [2013]. *MongoDB: The Definitive Guide*. 2nd Edition. O'Reilly, May 2013. 430 pages. ISBN 1449344682 (cited on page 30).

Express [2019]. *Express*. https://expressjs.com/ (cited on page 27).

Facebook [2019]. *React: A JavaScript Library for Building User Interfaces*. 07 Sep 2019. https://reactjs.org/ (cited on page 32).

Fain, Yakov and Anton Moiseev [2018]. *Angular Development with TypeScript*. 2nd Edition. Manning, 17 Dec 2018. 560 pages. ISBN 1617295345 (cited on page 31).

Fedosejev, Artemij [2015]. *React.js Essentials*. Packt Publishing, Aug 2015. 208 pages. ISBN 9781783551620 (cited on page 32).

Fenton, Steve [2018]. *Pro TypeScript: Application-Scale JavaScript Development*. 2nd Edition. Apress, 2018. 287 pages. ISBN 9781484232491 (cited on page 27).

Freeman, Adam [2017]. *Essential Angular for ASP.NET Core MVC*. Apress, 26 Jul 2017. 320 pages. ISBN 1484229150 (cited on page 31).

Gaffney, Gerry [2001]. *Structure Evaluation*. Information & Design. 2001. https://web.archive.org/web/20030421194101/http://infodesign.com.au/ftp/StructureEvaluation.pdf (cited on page 7).

Gaffney, Gerry [2006]. *Classified*. Information & Design. 27 Oct 2006. http://web.archive.org/web/20071101005502/http://www.infodesign.com.au/usabilityresources/classified/default.asp/ (cited on pages xi, 9–11).

Giamas, Alex [2019]. *Mastering MongoDB 4.x*. 2nd Edition. Packt, 30 Mar 2019. 394 pages. ISBN 1789617871 (cited on pages 30–31).

GitHub [2019]. *GitHub*. 07 Sep 2019. https://github.com/ (cited on page 28).

Google [2019a]. *Angular*. 27 Sep 2019. https://angular.io/ (cited on page 31).

Google [2019b]. *Attribute Directives*. 07 Sep 2019. https://angular.io/guide/attribute-directives (cited on page 32).

Google [2019c]. *React, Vue, Angular – Explore*. Google Trends. 07 Sep 2019. https://trends.google.com/trends/explore?cat=733&date=today%205-y&q=React,Vue,Angular (cited on page 34).

Haverbeke, Marijn [2018]. *Eloquent JavaScript: A Modern Introduction to Programming*. 3rd Edition. No Starch Press, 04 Dec 2018. 472 pages. ISBN 1593279507. https://eloquentjavascript.net/ (cited on page 26).

Haviv, Amos Q. [2016]. *MEAN Web Development*. 2nd Edition. Packt Publishing, 30 Nov 2016. 305 pages. ISBN 1785886304 (cited on page 35).

Hughes-Croucher, Tom and Mike Wilson [2012]. *Node: Up and Running*. O'Reilly, Apr 2012. 200 pages. ISBN 1449398588 (cited on page 26).

IBM [2019]. *MEAN Stack*. https://ibm.com/ (cited on page 35).

Jakobus, Benjamin [2018]. *Mastering Bootstrap 4*. 2nd Edition. Packt, 22 Feb 2018. 354 pages. ISBN 1788834909 (cited on page 28).

jQuery [2019]. *jQuery*. 07 Sep 2019. https://jquery.com/ (cited on page 28).

Keith, Jeremy [2010]. *DOM Scripting: Web Design with JavaScript and the Document Object Model*. 2nd Edition. Apress, 27 Dec 2010. 314 pages. ISBN 1430233893 (cited on page 26).

Keith, Jeremy and Rachel Andrew [2016]. *HTML5 for Web Designers*. 2nd Edition. A Book Apart, 2016. 90 pages. ISBN 1937557243 (cited on page 25).

Le, Thai, Shomir Chaudhuri, Jane Chung, Hilaire Thompson, and George Demiris [2014]. *Tree Testing of Hierarchical Menu Structures for Health Applications*. Journal of Biomedical Informatics 49 (Jun 2014), pages 198–205. ISSN 1532-0464. doi:10.1016/j.jbi.2014.02.011. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4058398/pdf/nihms570881.pdf (cited on page 6).

Macrae, Callum [2018]. *Vue.js: Up and Running: Building Accessible and Performant Web Apps*. O'Reilly, 23 Mar 2018. 158 pages. ISBN 1491997249 (cited on page 33).

Marquis, Mat [2016]. *JavaScript for Web Designers*. A Book Apart, 2016. 133 pages. ISBN 1937557464 (cited on pages 25–26).

Martin, Lisa Maria [2019]. *Everyday Information Architecture*. A Book Apart, 2019. 126 pages. ISBN 193755774X. https://everydayia.com/ (cited on page 3).

Mean.js [2019]. *Mean.js*. https://meanjs.org/ (cited on pages 27, 35).

Microsoft [2019]. *TypeScript*. 02 Oct 2019. https://typescriptlang.org/ (cited on page 27).

Mikowski, Michael S. and Josh C. Powell [2013]. *Single Page Web Applications: JavaScript End-to-End*. Manning, 30 Sep 2013. 432 pages. ISBN 1617290750 (cited on page 31).

MongoDB [2019a]. *Flexible Enough to Fit Any Industry*. 07 Sep 2019. https://mongodb.com/who-uses-mongodb (cited on page 30).

MongoDB [2019b]. *MongoDB*. 07 Oct 2019. https://mongodb.com/ (cited on page 30).

Mozilla [2019a]. *CSS Grid Layout*. https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Grid_Layout (cited on page 29).

Mozilla [2019b]. *HTTP*. https://developer.mozilla.org/en-US/docs/Web/HTTP (cited on page 25).

Node [2019]. *About Node.js*. 07 Sep 2019. https://nodejs.org/en/about (cited on page 27).

O'Brien, Dave [2017a]. *Tree Testing for Websites*. 07 Apr 2017. https://treetesting.atlassian.net/ (cited on page 5).

O'Brien, Dave [2017b]. *Tree Testing for Websites: Analyzing results*. 07 Apr 2017. https://treetesting.atlassian.net/wiki/spaces/TTFW/pages/1310944/12+-+Analyzing+results (cited on page 22).

OBS Studio [2019]. *OBS Studio*. 07 Oct 2019. https://obsproject.com/ (cited on page 49).

OW [2019]. *Treejack*. Optimal Workshop. 07 Sep 2019. http://treejack.com/ (cited on pages 6, 17).

PlainFrame [2014]. *PlainFrame*. https://web.archive.org/web/20140226221728/http://uxpunk.com/plainframe/ (cited on page 13).

Robbins, Jennifer [2018]. *Learning Web Design*. 5th Edition. O'Reilly, 21 May 2018. 808 pages. ISBN 1491960205 (cited on page 25).

Rosenfeld, Louis, Peter Morville, and Jorge Arango [2015]. *Information Architecture: For the Web and Beyond*. 4th Edition. O'Reilly, 11 Oct 2015. 486 pages. ISBN 1491911689 (cited on page 3).

Salesforce [2019]. *Create a Web App and RESTful API Server Using the MEAN Stack*. 09 Oct 2019. https://devcenter.heroku.com/articles/mean-apps-restful-api/ (cited on page 57).

Schilb, Steffen [2006]. *Design, Implementation, and Evaluation of a Tool for Testing Hierarchy-Based Web Navigation Systems*. Master's Thesis. University of Bremen, 03 Sep 2006. 148 pages. http://steffenschilb.com/Master_thesis_Steffen_Schilb.pdf/ (cited on pages xi, 3, 5, 11–13).

Schilb, Steffen [2014]. *C-Inspector*. 07 Sep 2014. http://c-inspector.com/index.php (cited on pages xi, 11, 14).

Spencer, Donna [2009]. *Card Sorting: Designing Usable Categories*. Rosenfeld, 05 Apr 2009. 162 pages. ISBN 1933820020. http://rosenfeldmedia.com/books/card-sorting/ (cited on page 3).

Spencer, Donna [2014]. *A Practical Guide to Information Architecture*. 2nd Edition. UX Mastery, 2014. 421 pages. ISBN 9780992538026 (cited on pages xi, 3–4, 7, 11).

TUG [2019a]. *TU Graz*. Graz University of Technology. 07 Sep 2019. https://tugraz.at/ (cited on pages 7, 49).

TUG [2019b]. *TUGRAZonline*. Graz University of Technology. 07 Sep 2019. https://online.tugraz.at/ (cited on page 7).

UZ [2019]. *UserZoom: User Experience Research Platform*. 07 Sep 2019. https://userzoom.com/ (cited on pages 6, 15).

Volkside [2019]. *Naview*. 07 Sep 2019. https://naviewapp.com/ (cited on page 13).

Vue [2019]. *What is Vue.js?* 07 Sep 2019. https://vuejs.org/v2/guide (cited on page 33).

W3Techs [2019a]. *Usage of JavaScript Libraries for Websites*. 07 Sep 2019. https://w3techs.com/technologies/overview/javascript_library/all (cited on page 28).

W3Techs [2019b]. *Usage Statistics and Market Share of Bootstrap for Websites*. 07 Sep 2019. https://w3techs.com/technologies/details/js-bootstrap/all/all (cited on page 28).

Wise Coders [2019]. *DbSchema*. 12 Oct 2019. https://dbschema.com/ (cited on page 36).