FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Pattern Based Usability Testing

Francisco Carvalho Rodrigues

PREPARAÇÃO DA DISSERTAÇÃO

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Cristina Ramada Paiva

July 23, 2018

Pattern Based Usability Testing

Francisco Carvalho Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

July 23, 2018

Abstract

Everyday we interact with all kinds of software through the use of Graphical User Interfaces (GUI's), and web applications are no exception. Since the GUI is usually a large piece of the software itself, it should be tested accordingly.

When testing GUI's, some behaviors and issues are common to a vast majority of systems thus the testing of these features should be identical, making it possible to identify patterns in these processes. As a result pattern based solutions such as the Pattern Based GUI Testing (PBGT) were created as an effort to automate generic solutions for these recurring problems.

Usability Testing is a method to evaluate the ease-of-use, intuitiveness and ergonomics of a system, regarding the interaction with its users. Typically, this is a task that is very dependent of human input relying on several stakeholders to be completed, therefore the results are in some way influenced by the subjectivity of the evaluators.

Being that the observations made regarding patterns in GUI testing can also be applied to Usability testing, the goal is to study and implement Usability Testing solutions with the same Pattern Based principles, by identifying usability patterns, implementing their systematic solutions and applying them in real scenarios measuring the accuracy of these solutions regarding their ability to test the desired patterns.

The Pattern Based Usability Testing solution provides an automated solution for testing an array of usability test cases that are common to many of the existing and future web applications. The usage of this tool should be as a complement to the conventional usability testing, providing beforehand objective testing results that are sure to be useful in the following process, increasing its efficiency and reducing the need for human involvement.

Resumo

Todos os dias interagimos com inúmeros tipos de *software* através da utilização de *Graphical User Interfaces* (GUI's), e as aplicações *web* não são excepção. Como, normalmente, a GUI constitui uma grande parte do *software*, esta deveria ser testada adequadamente.

Quando se testam GUI's, alguns comportamentos e problemas são comuns a uma grande maioria dos sistemas, como tal, o processo de teste destas funcionalidades deveria ser idêntico, tornando possível a identificação de padrões nestes processos. Como resultado, soluções de teste tais como o *Pattern Based GUI Testing* (PBGT) foram criadas como uma tentativa de automatizar soluções genéricas para estes problemas recorrentes.

O teste de usabilidade é um método para avaliar a facilidade de utilização, intuitividade e ergonomia de um sistema, relativamente à interação com os seus utilizadores. Tipicamente, esta é uma tarefa muito dependente do *input* humano fiando-se em diversos atores para ser completada, consequentemente os resultados são, de certa forma, influenciados pela subjectividade dos avaliadores.

Considerando que as observações feitas relativamente a padrões no teste de GUI's, estas podem também ser aplicadas a testes de Usabilidade, o objectivo é estudar e implementar soluções de teste de Usabilidade com os mesmos princípios baseados em padrões, identificando padrões de usabilidade, implementando as suas soluções sistemáticas e aplicando-as em situações reais medindo a precisão destas soluções a respeito da sua capacidade de testar os padrões escolhidos.

A ferramenta de teste *Pattern Based Usability Testing* oferece uma solução automatizada para testar um conjunto de casos de teste de usabilidade que são comuns a muitas das ferramentas *web* atuais e futuras. A utilização desta ferramenta deve ser como um complemento aos testes de usabilidade convencionais, fornecendo, de antemão, resultados de teste objectivos, úteis para o processo que se segue, aumentando a sua eficiência e reduzindo a necessidade de envolvimento humano.

Acknowledgements

It is with a bittersweet feeling that I finalize this period of my life with the conclusion of this work and I am surely happy to share the accomplishments with the people that stood beside me over the course of these years.

To my family, Ki and Berto, Diogo and Rita, Vó Manecas and Vô Mando, cousins and uncles, thank you for supporting me in every way imaginable and for putting up with me since I was an annoying brat.

To Carolina, this dissertation is almost as much yours as it is mine, thank you for being there in the best and the worst of times.

To my friends, I am only thankful for meeting you guys over the course of this journey, the support and the good times is just what we do. Special mention to Daniel Nunes for all the "Are you working?" that kept me in line.

To the AEFEUP family, thank you for making me grow day after day and for all the adventures surrounded with awesome individuals. Especially Luís Natividade and Abel Tiago, the best companions I could ask in the most rewarding experience I had in these years.

To professor Ana Paiva, thank you for the valuable guidance and advice that allowed me to accomplish this work.

To everyone that I failed to mention in this brief summary of what could be a full-fledged book, you are not forgotten.

Francisco Rodrigues

"That's all, folks."

Looney Tunes

Contents

List of Figures

LIST OF FIGURES

List of Tables

Abbreviations

GUI	Graphical User Interface
PBGT	Pattern Based GUI Testing
PBUT	Pattern Based Usability Testing
SUT	System Under Test
URL	Uniform Resource Locator
XML	Extensible Markup Language
XPATH	XML Path Language
HTML	Hypertext Markup Language

Chapter 1

Introduction

This dissertation is entitled "Pattern Based Usability Testing" and aims to find, study and implement usability testing patterns. It falls under the category of Software Engineering, more specifically Software Validation and Verification.

1.1 Context

With the advent of customer-end technology, the interaction of the user with the system started to gather more and more attention, therefore Graphical User Interfaces, responsible for this kind of interaction, naturally started growing both in functionality and complexity. As a consequence, in nowadays software, it is common for the GUIs to represent more than half of the whole system's code [Mem02], thus being even more crucial to test them appropriately.

In the design of many different GUIs it is possible to find some functionalities that solve similar problems and are similar in objectives and behavior. Finding these patterns that portray recurring solutions for common design problems is a practice that can save time when testing GUIs and is a good starting point in introducing automation in this discipline of testing. The Pattern Based GUI Testing project explored this approach by identifying patterns and finding generic solutions for them capable of testing the same behavior in several systems achieving a higher degree of systematization and automation when performing the testing of GUIs.

The usability testing practice is a particularly subjective one [JHJ98], since it is easier to notice bad usability rather than a good one, its a difficult job to evaluate it. The process typically consists in a group of users, guided by an experienced individual that try to find flaws in the usability of the system under test, following a walkthrough based on a set of heuristics and although there are some tools that can aid in this process they work more as a complement rather than a substitute, and offer a small degree of automation to the testing process itself.

1.2 Motivation

As the relevance of GUIs is increasing so is the importance of testing them in all aspects possible. Usability is one of the many characteristics that can be tested to assure the quality of the GUI and subsequently the system's. By introducing some degree of automation in the Usability Testing process it is expected to increase the independence relative to the human involvement and subjectivity improving the time-efficiency as well as lowering the human induced error rate.

The Pattern Based approach carried out by the PBGT is a good way of extending the reach of the tests developed, since it tests a wide range of systems that employ the designs covered by the implemented patterns, following the same approach to implement usability patterns, that can potentially extend the PBGT platform for usability testing, provides the same advantages for a different branch of testing, therefore the implementation of a Pattern Based Usability Testing solution provides a new level of testing complimentary to the PBGT reaching for a bigger depth and providing a different angle in the testing of GUIs, not acting as replacement for the traditional methods, but as a mechanism to improve it in coexistence, bringing to the table the advantages of automation, a more thorough testing, and early exposure to failures that could otherwise go unnoticed.

In the field of usability testing, the the methods that are currently put in practice rely considerably in the manpower of evaluators and testing experts. This fact is due to the recurring challenge of achieving a high degree of automation regarding this aspect of the systems under test, since usability is closely tied to the user experience itself, it's not clear how to get a machine to perceive the system in the same way as the humans who make use of it. As a result, automated testing related with usability must be carefully planned since it is fairly easy to infer wrong conclusions when resorting only to algorithms for the evaluation of usability. A good approach to this problem would be to gradually increase the degree of automation without discarding the human interaction from the testing process, harmonizing the two approaches into a more robust one. This can be done by implementing solutions that gather information regarding usability and process this information in a way that can provide some knowledge about the system's usability to the evaluator as a starting point as well as a way to steer the course of the evaluation itself into more articulate conclusions.

1.3 Goals

The main goals to be achieved during the course of the development of this dissertation are:

- **Research and Identify Usability Patterns** - What usability patterns are already cataloged, determine what can be considered a usability pattern and what unidentified patterns may exist.
- **Analyze and Implement** - Take the information gathered in the previous stage followed by a recognition of their implementability, utility and scope in order to decide what patterns should be implemented in the context of the project. Thereby proceeding to the implementation itself.
- **Validate the implemented patterns** - Setting up several case study scenarios with systems customary to the average user, and executing the implemented tools with focus on the results provided by them and whether they are accurate, pertinent and useful.

1.4 Dissertation Structure

Other than the present introductory chapter, this dissertation is composed by five additional chapters.

- The second chapter [2] contains the description of the state of the art relative to usability testing, exploring the methods currently used as well as tools correlated to this module of testing.
- The third chapter [3] presents an overview of the problem and a high level explanation of the proposed solution in a more theoretical fashion.
- The fourth chapter [4] gives a lower level view of the solution, focusing more on the implementation itself and exploring more thoroughly its details.
- The fifth chapter [6] consists of the case studies used to evaluate the results of the solution, presenting outcomes of its testing.
- The sixth chapter [] concludes the dissertation providing a summary of the work done and the insights on the accomplishment of the objectives and potential future work.

Chapter 2

Usability Testing

This chapter includes the description of the state of the art and existing work in the domain as well as a technical review to technologies in the field of Usability Testing.

2.1 Introduction

When testing a system for usability there are four ways of evaluating this aspect: automatically, empirically, formally and informally [Nie94]. In the current scenario, automatic usability testing methods are still hard to implement and to get them to cover the basic requirements needed for a complete usability evaluation and formal methods are also hard to apply and to scale for large SUTs. Therefore, in this chapter will be explored informal and empirical methods as well as technological tools that are useful in any of this methods.

2.2 Informal Usability Testing

This section includes every way of evaluating that doesn't rely on any conventional procedures, giving more focus to the feedback of the users, rather than how it is obtained. In Informal Usability Testing there is no need for trained individuals, apart from the facilitator. It's an ampler approach in order to get more input, faster than with its formal counterparts. [Spi07]

The format is flexible and not well defined, although typically it consists in a walkthrough usability test with several participants in the same room, focusing some issues already found and areas more prone to failure. It should be asked to the participants to talk freely about their thoughts of the system and to pinpoint some points where improvement might be needed. [Wan16]

2.2.1 *Quick-and-dirty Method*

Quick-and-dirty is a particular usability inspection method. Its a common practice and can be useful since not all data needs to be gathered using formal methods. It is a process where the GUI designers gather feedback from the users to check if their ideas are aligned with the users' needs, interfaces are discussed with colleagues and other developers, and the users are asked to report errors and provide suggestions in a casual fashion and with no formal record of the results, which is especially useful since it can be done at any instant and gives quick insights whenever needed.

2.3 Empirical Usability Testing

2.3.1 User Studies

If our objective is to know how the user will react to the system, a natural solution will be to directly reach to them and collect their response. This can be done by inquiring them or observing the direct interaction with the system, thereon leading to the existence of several methods.

2.3.1.1 Surveys

Collecting data by surveying the users is certainly a fast way to reach a considerable crowd and obtain large chunks of information. [Dum03] Although there is no difficulty in retrieving the information, drafting a good survey is not that much of an easy task and requires experience to be designed in a way that makes it feasible to make sense of the information.

Despite having the capability of providing useful information and proving to be valuable together with other methods, a true measure of usability cannot be perceived through the use of surveys, thus its standalone performance when testing usability is questionable.

2.3.1.2 Focus Group

Conventional focus groups consist in gathering a group of users in a room to discuss the usability of the SUT, a usability specialist acts as a moderator of the meeting steering the conversation to relevant topics, annotating relevant observations and stimulating the discussion meanwhile trying not to affect the final results of the talk.

Focus Groups are believed to be a valuable method for gathering usability data when properly devised and with a good moderator. Despite not being able to completely assess the usability of a system its results can bring about new ideas and validate current assumptions. Nevertheless the reliability of the system can be questioned because " ... the quality of the data obtained from usability focus groups is only as good as the quality of the participant selection and the questions asked." [RCC+02] and although the collective scenario can put the participants more at ease it can also create some biases since they can start to be influenced by the overall opinions leading to one consensual opinion rather than several disputing opinions that could be more of use.

2.3.1.3 Laboratory Usability Testing

This method is one of the most popular and effective, but also one of the most complex, it requires infrastructure and a great deal of planning in order to be executed with the intended results. It is designed with the intent of observing and recording the interaction of a user with the system.

The lab disposition should be focused on the system being tested and the user testing it with video cameras pointed at it and a one way mirror that allows the observers to watch the user behavior without him feeling the pressure of having someone analyzing their actions.

Some attention should be given to the evaluators choice, since they should represent the end user that the software targets, for this a profile of the users should be found so that it is known which characteristics to look for in a participant in order to achieve the best results. The ideal number of participants needed to find most of the usability flaws is a topic of discussion [Dum03], it is estimated that with 5 participants around 80% of the problems should be uncovered, with 10 participants this number would reach 90% and with each additional participant would increase it by an even smaller factor. Therefore no matter how many participants we choose we should know that not all usability problems will be found. [Dum03]

With the lab set up and the participants chosen, all that's left to start evaluating the system is the script, what activities should the user attempt to perform, in what order and with what context. These tasks should include usual actions that users perform often and tasks that approach areas prone to the existence of problems. For every task a context is established so that the user can relate to a real life scenario.

The testing itself occurs when the participant starts using the system following the script provided and describing out loud his perception of the interaction while his actions are being observed and recorded, both by the cameras and by auxiliary tools that register other behaviors of the user, for instance mouse clicks and trace, time to perform actions and eye tracking. All this data gathered from the observations is analyzed afterwards by the usability specialists that try to reach insights about the system and the changes that should be made.

This method is widely favored by specialists because:

- It is believed to uncover the flaws more likely to happen and the ones that are more critical
- The results can be used right away to improve the product
- It is conducted from the point-of-view of the user allowing the observations to be more neutral and unbiased

Although there are some cons to it as well, such as:

- Despite being performed to mimic the user's behavior, the scenario is fabricated, thus not echoing the reality in certain cases
- If the system passes the test it is not proven that it has good usability, simply that the assigned tasks have.

- Although the participants can portrait many of the characteristics of the target audience they are never a whole representation of the user base.

- The setting is expensive and the experience requirements regarding the facilitator and observers are high for the process to run smoothly

[SB97] [Dic02] [RRH00] [KCF92]

Granting that the laboratory method is indeed robust and efficient, it is not a method accessible to everyone and its not easy to implement and at the same time is very human dependent meaning that the results can be good or bad depending on the performance of those involved.

2.3.2 System Inspection

Instead of focusing on the users, the System Inspection methods analyze the system, they consist in having individuals experienced in usability who examine the product according to the ground rules of the method chosen.

From a logistic point of view these methods are easier to implement than User Studies and typically involve less testing effort, however the areas explored are quite different, while system inspection finds fields in need of more testing, user studies finds areas in need of adjustment. [Sav96]

2.3.2.1 Heuristic Evaluation

The Heuristic Evaluation method is another of the most popular evaluation methods mainly because it is fast and low-cost since all that is needed to implement it is one or more evaluators, the system itself, and a set of rules to follow called heuristics. [RRH00]

The process consists in having the evaluators judge the system's components' usability based on checklists that are put together following a set of established fundamentals. The fully finished product can be the object of this evaluation, but it is also possible to evaluate a product in development or even prototypes.

There are many sets of heuristics developed by scholars in the usability area, and there isn't one of them regarded as the most effective or the most complete, therefore it is always a matter of preference of the evaluators on which heuristics to use.

Whether or not the method is successful is dependent on the skill and expertise of the evaluator. "Worst" evaluators are more likely to miss more problems than the ones with more experience. Nevertheless even with a good evaluator one single heuristic evaluation is not as effective as having several evaluators doing the same work. It was found that various evaluations were consistently more effective than a single one. Thence the course of action that is normally followed is having assorted evaluators, each with their method explore the system freely and reach their own conclusions, afterwards all the results are gathered and compared to reach a unified assessment of the system's usability. [Nie92]

- **Nielsen's Heuristics** [Nie94]

  As stated before, there are numerous takes on finding the set of heuristics that are the most effective in finding the most and the more critical usability problems, although there is no consensus on the existence of a flawless set of rules, the heuristics devised by Nielsen seem to be the most widespread when it comes to the evaluators' preference.

	Heuristics [Nie05]
1	Visibility of system status
2	Match between system and the real world
3	User control and freedom
4	Consistency and standards
5	Error prevention
6	Recognition rather than recall
7	Flexibility and efficiency of use
8	Aesthetic and minimalist design
9	Help users recognize, diagnose, and recover from errors
10	Help and documentation

Table 2.1: List of Nielsen's heuristics

The names of the heuristics presented on the table above were designed to be descriptive and self-explanatory, however in order to understand the extent of each rule, it is necessary to explore other sub-heuristics that fall under the scope of the main one. For instance, "Visibility of system status" can be subdivided in heuristics such as "Show that input has been received", "Feedback timely and accurate" and "Feedback provided for all actions", and the same happens for every other heuristic, stating that there is more in studying a set of heuristics than knowing the rules themselves.

- **Shneiderman's Golden Rules** [Shn10]

  Another successful take on usability heuristics were Shneiderman's Golden Rules, that also became popular and proved to be useful when designing productive and frustration-free interfaces.

	Heuristics
1	Strive for consistency
2	Enable frequent users to use shortcuts
3	Offer informative feedback
4	Design dialogue to yield closure
5	Offer simple error handling
6	Permit easy reversal of actions
7	Support internal locus of control
8	Reduce short-term memory load

Table 2.2: List of Shneiderman's Golden Rules

2.3.2.2 Cognitive Walkthrough

Cognitive Walkthrough is a method that focuses the user's analytical process when using the system. The core of the process involves the evaluator trying to predict the user's actions, behaviors and thinking process.

Built on constructivist theories assuming that the user adopts a discovery learning behavior, the method centers itself on user actions and whether the system makes them easier or slows them down. Theoretically, the interaction between the user and the system is partitioned in four stages:

- The user establishes an objective he wants to achieve with the system
- The user scans the interface for possible interactions
- The user selects the option that seems the best to make way in the direction of the goal
- The user carries out the operation and evaluates the outcome provided by the system

Based on this principle the evaluators have to put themselves in the shoes of the user and anticipate what he would think in each stage for the determined task. These tasks are pre-defined and must be those that users are prone to perform making use of the system, the goals behind them should also be specified as well as sub-goals for the former. Afterwards the process likely to be used by the user to complete the tasks is identified by predicting flow of actions that he needs to execute in order to achieve his goal, the existence of various sequences of actions for the same task is possible. Thereupon the evaluators must follow these courses of action from the point-of-view of the user observing the systems response and making use of their experience and perception to identify eventual usability flaws.

The Cognitive Walkthrough method was found to be useful in usability testing, despite having some challenges tied to it. One of the main issues with the method is the evaluators' missing knowledge of the theory that can lead to misconceptions that are likely to affect the final results. The process can sometimes be a dreary task to the evaluators and result in decreases in motivation and subsequently in the accuracy of the evaluation. Since the process follows a highly detailed approach it is in some cases very time consuming. The performance of the method is only as good as the experience and skill of the evaluators either in the task specification as well as the action walkthrough. [**?**] [LPWR90]

2.4 Automated Tools

Like in every other testing approach, Usability Testing also has automated tools for its purpose. However, there is none that can automatize the whole process, neither replace the human involvement in the processes. Although these tools do not replace the traditional methods they can provide valuable information in order to greatly improve the final results.

Based on the research done, the tools related to usability testing fall under essentially three categories: User Interaction Analysis, A/B Testing and Layout Analysis that will be further explored in the following section.

2.4.1 User Interaction Analysis

When a user interacts with a system there are many observations we can make of its behavior that can be useful to make assumptions regarding the usability of the system. As an example, that is the reason why in Laboratory Usability Testing observers are placed across from the one-way mirror and cameras are set up to record the user. However there are aspects that can be relevant to the outcome of the testing process that cannot be translated to useful data simply by observing the user interact with the system.

With User Interaction Analysis tools it is possible to gather the kind of information about the user's interaction with a system that isn't obtainable with regular means. Examples of data that can be gathered with these tools are:

- Mouse clicks and keyboard input
- Mouse trace
- Eyetracking

These metrics if analyzed properly can offer great information when analyzing the usability of the system, some interesting metrics they can reveal are, for instance: what sections of the interface the user pays the most and the least attention, hesitations when completing an action, areas of the interface that are neglected. Although this information doesn't really tell which usability flaws exist in the interface, it points to some of them and with the right analysis skills interesting conclusions can be made.

2.4.1.1 Usaproxy

[AS07]

This tool allows to record the users' behavior in a non intrusive way, not changing the way the user interacts with the system, simply recording his actions. By monitoring and listing the user's actions it provides the evaluator with additional metrics for him to best decide the future steps to improve the system's usability.

The kinds of actions that Usaproxy records are:

- User navigation behavior - for example, switching between pages
- Mouse tracking - where the mouse clicked, elements hovered, absolute mouse position
- Time to complete actions - time user spends on page, time the mouse spends over items, click frequency
- Other actions - page scroll trace, interface resize, keys pressed

2.4.1.2 CrazyEgg

[Cra13]

CrazyEgg is a tool that records mouse events and presents the information in a user-friendly fashion. It generates four different kinds of maps:

- Heatmap - Visual representation of the frequency of the clicks in the web page, showing the contrast between areas that are more and less clicked.

- Scrollmap - Measures how far the user scrolls, the areas where he stops scrolling and when he leaves the page, allowing to make assumptions on what is capturing the user's attention and what is making the user leave the page

- Overlay - Gives the percentage breakdown of clicks the element of the page. Useful to determine which webpage calls-to-action are the most popular.

- Confetti - Presents a precise representation of individual clicks on the page displayed by color based on various metrics.

Figure 2.1: Example of Confetti map generated by CrazyEgg - Source: http://www.toolsinsight.com

2.4.1.3 EyeTracking

[PN09]

Eyetracking is a process for measuring the point of gaze of a person. The tools designed for usability testing can make use of this technology to assess the response of the user to stimuli and the focus of his attention when using an interface. From this information it is possible to know which areas of the interface are more appealing to the user, where should be placed the more relevant information and what distracting factors can be eliminated.

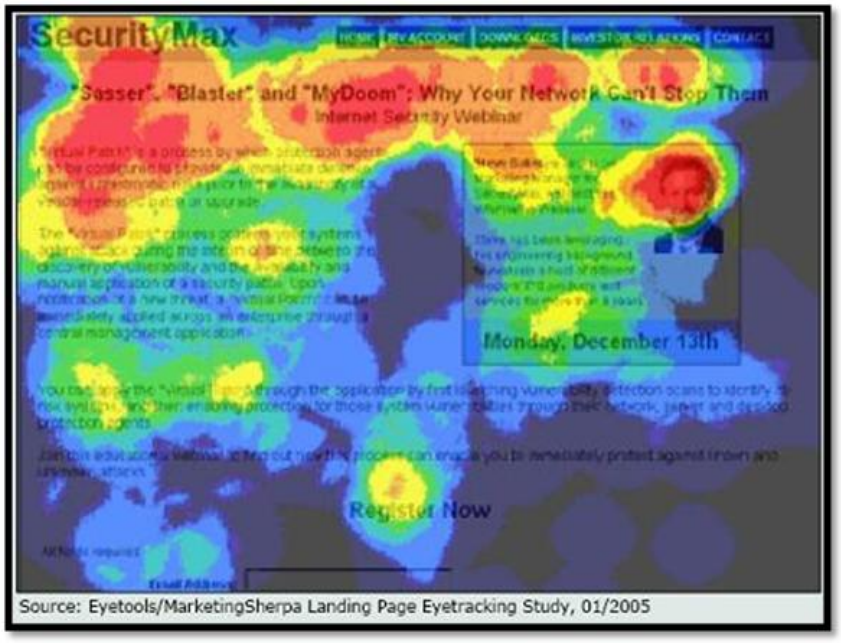

Figure 2.2: Example of a heatmap generated by an EyeTracking mechanism

2.4.2 A/B Testing

[Cho12]

A/B Testing is a method suitable to test usability that consists in comparing two versions of the same system with slight variations in order to determine which one is the most efficient. One of the versions is the *control* and the other one is the *variation* where the small change is inserted. The test is then conducted by presenting each version to half the test population and measuring the performance and the effect on the users regarding both versions. This data is then analyzed to understand if the changes made in the *variation* have a positive effect and should be added to the system.

The test process can be divided in six stages:

- Explore data from previous usability inspections in order to perceive where should be inserted possible changes
- Establish what metrics to use to evaluate the success of the variation

- Formulate hypothesis of possible solutions or improvements to the problems found on the system
- Create variations that have the potential to be better than the current existing features
- Release the test versions to the public where only half the population will visualize the variations
- Analyze the data gathered during the previous stage

This process can be very useful to improve small features and reach the best possible version of the interface, however it requires usability testing to be done in advance in order to know which problems to tackle, otherwise it would be extremely costly in time and resources to evaluate the whole system using this method.

2.4.2.1 AttrakDiff

The AttrakDiff tool uses A/B Testing to understand how the users perceive the usability of a system. It supports additional features other than A/B testing, such as, individual evaluations of the system based on the participation of users.

When it comes to A/B testing, this tool allows to evaluate the two versions of the interface, control and variation, separately and by comparison. It uses a questionnaire to assess the pragmatic and hedonic of both versions, and presents the results afterwards for posterior analysis.

Figure 2.3: AttrakDiff classification system - Source: attrackdiff.de

2.4.2.2 Optimizely

Optimizely is designed specifically for A/B Testing, enabling to manipulate the original interface to create alternative versions of it in a fast and easy way. In posterior phases it gathers the information provided by the users' feedback and organizes the results by filters of information based on several criterion.

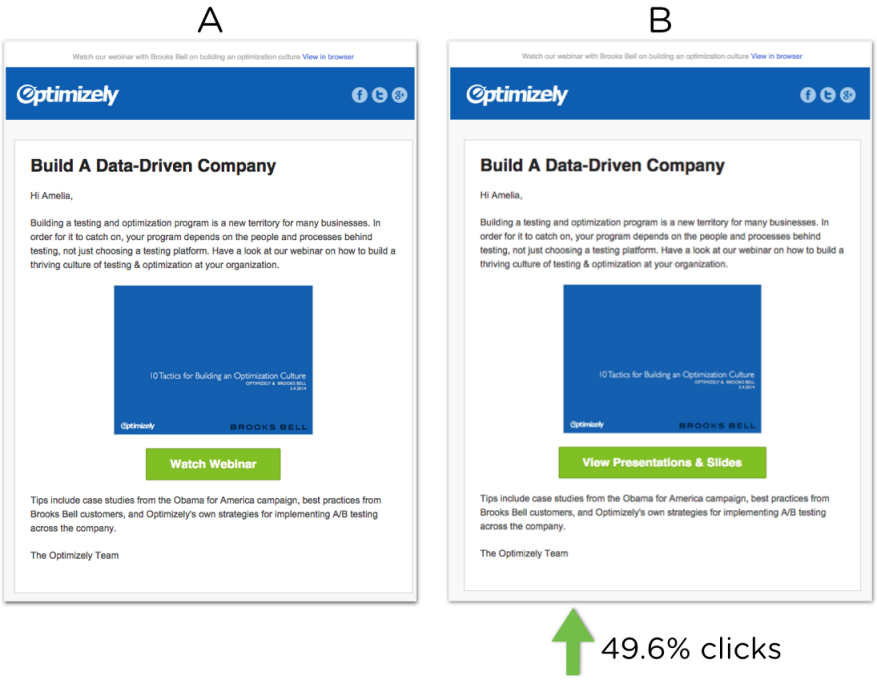

Figure 2.4: Example of A/B Testing - Source: blog.optimizely.com

2.4.3 Layout Analysis

This type of tools is based on quantitative metrics to evaluate the interface of web pages based on static analyses performed directly upon the page code. The data gathered by applying this process is directly linked with the disposition of the elements on the interface, some information that can be retrieved is, for instance: page density, element size comparison, alignment and color match.

Based on data provided by these tools it is possible to assume usability flaws, some that could be missed by an evaluator and others that require a more thorough vision than that of the human eye.

2.5 Pattern Based GUI Testing

Pattern Based GUI Testing is a model based testing methodology aiming to achieve automation and systematization in the testing process of Graphical User Interfaces [CPFA10]. The PBGT Tool follows this approach providing a graphical modeling and testing environment in order to test recurring behaviors in GUIs, therefore it implements testing patterns with the ability to test many different systems [AP18] in a similar way. It is built using the PARADIGM language, a Domain Specific Language to describe and analyze instances of patterns.

Since in the field usability there can also be found recurring behaviors, model based testing could also be an interesting approach for this kind of testing, as explored in [DP17].

The tool is deployed as an extension to the Eclipse environment on top of the Eclipse Modeling Framework [MP14d] [MPM13].

The PBGT Tool has five core components, one of them being the PARADIGM language among the other four presented below [MPNM17]:

- **PARADIGM-TE** is the component responsible for the test execution, analyzing the test coverage [PV17] and providing the test results and information related to them.

- **PARADIGM-ME** is the modeling environment component that enables the user to assemble and set up the test models.

- **PARADIGM-TG** is the test generation component that produces the test cases based on the models defined.

- **PARADIGM-RE** is the reverse engineering component that has the objective of deriving the PARADIGM models from the systems to be tested [SP14].

The testing process using this tool encompasses five phases which are: modeling, configuration, test case generation, test case execution and result analysis. The modeling phase can be done by the user, or using the reverse engineering tool.

The PBGT tool already has several implemented GUI test patterns, and this list can always be incremented if more patterns are added. Although the approach followed by the PBGT project is not usability oriented there is always the possibility to extend it in that way, since none of the existing patterns within the tool test the system for usability aspects, the extension would encompass identifying and implementing interesting aspects in the field of usability following the same approach, therefore extending the tool and broadening its spectrum of action.

2.6 Summary

It was possible to understand that usability testing is still very much dependent on the human involvement, being users, analysts, evaluators or observers. There are many variables added to the test process that are human induced and therefore subjective. The Evaluator Effect studies give insights on this matter, that the experience and skill of those involved directly influence the results and the success of the whole process.

One way of minimizing the human subjectivity in the process is trying to automatize some parts of it, since automating the whole process is still a distant scenario. Automation can come as a provider of new data that would otherwise be inaccessible by the sole human involvement, as a substitute for certain tasks otherwise performed manually (saving resources and time in the process) or as an additional level of testing. For the first two some of the tools studied already explored those concepts and added something to the usability testing approach, the Pattern Based Usability Testing aims to target the third.

By understanding what is currently done in the field along with the methods practiced in usability testing, a better overview of where the usability flaws are present was given, as well as what are the best ways, in each situation, to identify them. With a clearer view of what usability is and how it can be addressed, the perception that there is really a need for new approaches regarding usability was strengthened. The automated methods studied are certainly a first step in the right direction, however an additional effort must be made, and the implementation of usability patterns offers a new and promising approach.

Chapter 3

Pattern Based Approach to Usability

Over the course of this chapter it is described, in the first place, the study related to Usability Patterns that was performed, followed by the theoretical basis for the implementation choices that were made regarding the patterns, and the architecture of the solution and the tools used in its extend.

3.1 Test Patterns

A test pattern can be described in many different ways, some crucial aspects to mention in its description are elements such as *why* it should be adopted, *when* it should be used and *what* is its objective. With this purpose in mind, the pattern can thereat be defined following the subsequent format [MD97]:

- **Name**: the identifier of the pattern
- **Context**: the circumstances with which the problem occurs
- **Problem**: the problem taken in consideration by the pattern
- **Forces**: aspects taken in consideration that serve as motivations for the solution
- **Solution**: the description of the prospective solution for the problem (described further in this section)
- **Known Uses**: situations where the pattern can be applied
- **Example**: objective scenarios of the application of the pattern

According to [MP14a], related to the Pattern Based GUI Testing project, a Test Pattern can be defined as a tuple **< Goal, V, A, C, P >** its elements corresponding to [MP14c] [MP14b]:

- **Goal** is a descriptive ID of the pattern in question;
- **V** is a set of pairs variable, data depicting the relation of the variables involved in the test and its input values (provided by the user);
- **A** is the chain of actions to perform when executing the test case;
- **C** is a set of the verifications (or checks) to be executed after the actions took place;
- **P** is the precondition required for the execution of the test case to be possible.

This representation ultimately translates to a sequence of events meaning that for each **Goal**, if the **P**recondition proves to be true, the set of **A**ctions is performed with the input corresponding to the **V**alues, after the execution of the actions, the **C**hecks are verified.

While this definition was thought for patterns that test recurring behaviors in GUIs, it can be adapted to usability patterns, therefore further in this chapter this definition will be used to define the usability patterns explored in the context of this dissertation.

3.2 Usability Patterns

In the time researching the past work regarding usability it was noted that there wasn't much groundwork done regarding the definition and identification of usability patterns. Therefore a deeper analysis had to be made on what can be qualified as a usability pattern, what patterns can be found that fall in this category and which of those were appropriate to be implemented in this context.

3.2.1 Usability Pattern vs Usability Practice

As mentioned in the previous chapter, there are several sets of usability heuristics and rules [Nie05] [Shn10], all with the aim to improve the effectiveness and ease of use of the systems to give a better experience to the user.

When trying to identify the usability problems that can be the background for a pattern, resorting to the research of previously studied usability practices seemed like a good approach, since many of the good usability practices can reveal recurring usability problems, and other problems might already be identified themselves.

By following this approach it was reached the conclusion that many of the usability practices could not be translated to usability patterns (or at least the association wasn't obvious) lets take as an example the guideline "*Keep a minimalist design*", it's not an easy task to predict how minimalist a page is, or whether it is minimalist enough since it is not an objective concept thus being something that can be evaluated much more efficiently by a human evaluator.

Other usability principles although being viable and possible to implement could pose interesting decision-making challenges, for instance, a usability principle says "The more relevant the information, the closer it should be to the top-left corner" [Fes17]. In this case, the challenge

would be on how to decide how relevant was a piece of information relatively to another. And even if we could provide the tool with a hierarchy of relevance of the page elements, in a dynamic system such as a news portal or a social network it would only be as good as the page remained the same.

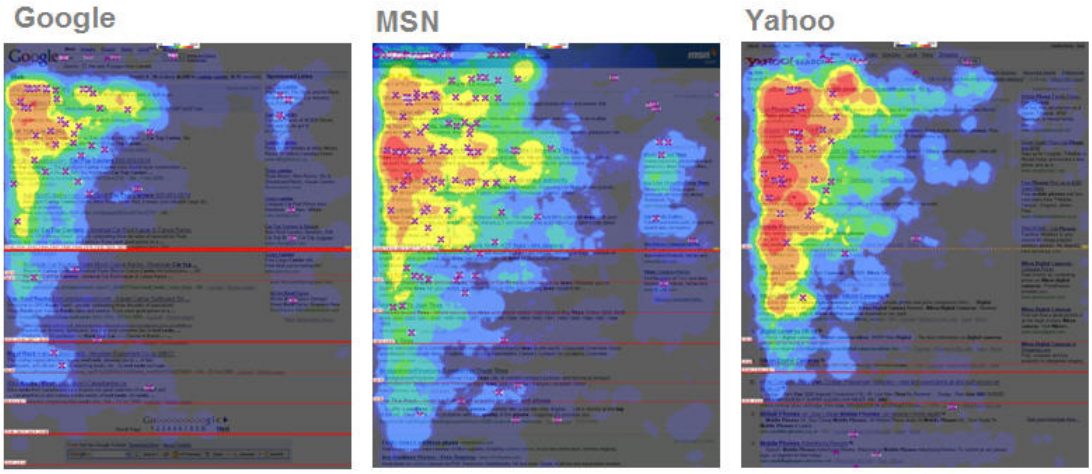

Figure 3.1: Eyetracking study showing user focus on the top/left corner information - Source: https://www.redalkemi.com

After exploring possibilities on what patterns could emerge and if they were reasonable in this context, implementable and above all useful, at last was decided to explore patterns related to element consistency, and error prevention since these usability aspects showed good perspectives of finding interesting approaches.

3.3 Consistency Usability Patterns

According to Nielsen [Nie94] in a factor study on which usability heuristics would better explain actual usability issues, consistency ranks first for all the usability problems in general and second for usability problems considered serious, thus verifying the importance of this usability principle.

In usability, consistency is related to how the system elements are presented to the user, meaning that elements that have the same objective and/or function in a similar way should be presented in a similar way and different elements should be made distinctive.

A system with good consistency has the advantage of improving the users' learning curve on how to use the system [Nie88] , and once the user learns how to navigate though it he is saving time in every interaction, this happens because recall is more efficient than recognition [Nie05]. Additionally, it minimizes the chances of getting the user confused, therefore reducing chances of frustrating situations happening.

Below are defined the explored usability patterns that fall under the category of 'Consistency'.

3.3.1 Element Layout Consistency Pattern

- **Context**: Elements in a web page should have similar layouts when serving the same purpose or having the same function.
- **Problem**: How to provide the user with information that can accurately be used to predict consistency between elements.
- **Forces**: Should give reliable measures in order to take conclusions regarding layout consistency.

  Should give the user the freedom to choose what parameters are relevant for the solution
- **Solution**: Provide a solution which gives information relative to chosen elements independently of the system where they are included
  - **Goal**: Element Comparison Output
  - **V**: {elements under test, user configurations}
  - **A**: [exploration process, element information gathering, information analysis and comparison]
  - **C**: {style difference between elements must be less than the sensivity defined in the preferences, elements' distance from an axis must not be bigger than the offset defined}

  During configuration, the user must provide one or more URLs and the locators of the respective elements desired to be tested (id, name, class or xpath). Must also set the user preferences, consisting in checking/unchecking the "CSS", "Size" and "Position" checkboxes depending on what is pertinent to analyze, setting a pivot element it being the standard for the comparison with the other elements, setting the CSS comparison sensitivity desired and enabling or disabling the checks. The objective of the solution is not to provide a verdict but instead to give the user an informative output.
- **Known Uses**:
  - Analyze several elements from different pages of the same system supposed to be consistent
  - Analyze elements implemented by different developers
- **Example**:
  - **V**: { [url, https://sigarra.up.pt/feup/pt/web_page.inicial], [element1, id="conteudoinner"], [element2, class="ultimas_noticias], [CSS, true], [Size, true], [Position, true] ,[pivot, 1], [sensivity, 70], [areaRatio, true], [dimensionDifference, false] }
  - **A**: [exploration process, element information gathering, information analysis and comparison]

  - **C**: { [element1 and 2 CSS similarity, false], [element 1 and 2 alignment, false] }
  - **P**: true

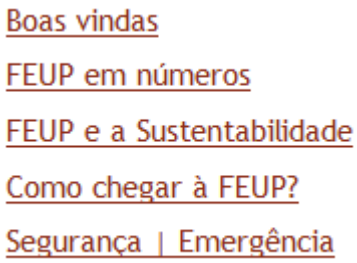

Figure 3.2: Example of elements consistent in style with vertical alignment

3.3.2 Page Text Consistency Pattern

- **Context**: The text presented in a determined page should be similar according to its purpose, for instance text with the same significance should have the same size and color on the other hand a title should be considerably bigger than the rest of the text.
- **Problem**: How to find and group the text elements according to their text style attributes
- **Forces**: Should identify text elements with the same properties

  Should group the elements according to their similarity
- **Solution**:
  - **Goal**: Element grouping and style attributes output
  - **V**: { page url }
  - **A**: [interaction with page, element information gathering]
  - **C**: { }
- **Known Uses**: Any text-based web page
- **Example**:
  - **V**: { [url, https://sigarra.up.pt/feup/pt/web_base.gera_pagina?P_pagina=1182] }
  - **A**: [provide url]
  - **C**: { }
  - **P**: true

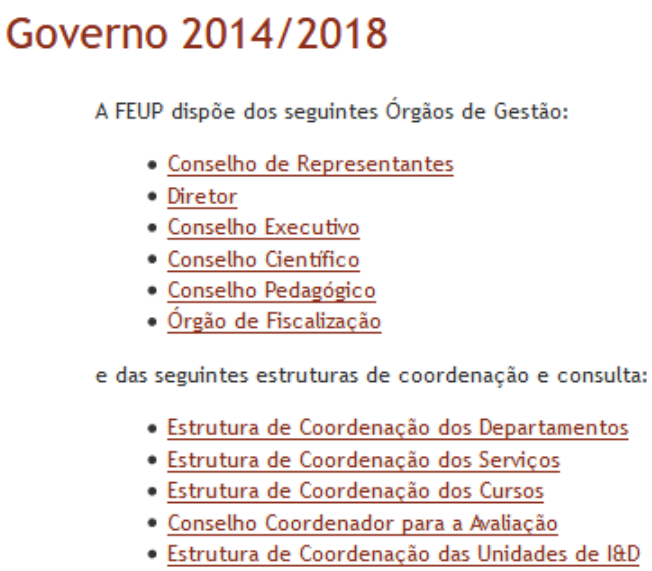

Figure 3.3: Example of text elements with a visual hierarchy

3.4 Error Prevention Usability Patterns

Providing the user with carefully thought and well designed error messages is certainly a good usability practice, however it would be even better if the error could be avoided at all, and that is where Error Prevention comes in. When your text editor corrects an error automatically, or when your email client asks you if you have forgotten the attachment, those are good examples of common error prevention applications. And since this is a relevant and helpful usability aspect it was also chosen to be further explored.

3.4.1 Tooltips and Placeholders Pattern

- **Context**: One way to prevent user-induced errors is to provide as much information as possible about the tasks he should perform, tooltips and placeholders in input elements are a common way of doing it.
- **Problem**: Verifying the presence of tooltips and placeholders and its values for a set of input elements.
- **Forces**: Should find existing tooltips and placeholders

  Should find the values of the tooltip and placeholder in case of their existence
- **Solution**:
  - **Goal**: Tooltip and Placeholder Values Output
  - **V**: {page URLs, elements under test}

  - **A**: [exploration process, element information gathering]
  - **C**: { Presence of tooltips and presence of placeholders }

- **Known Uses**:
  - Forms
  - Text entry fields

- **Example**:
  - **V**: { [url, https://sigarra.up.pt/feup/pt/web_page.inicial], [element1, id="user"], [element2, id="pass"] }
  - **A**: [provide url]
  - **C**: { [placeholder1, true], [placeholder2, true], [tooltip1, true], [tooltip2, true] }
  - **P**: true

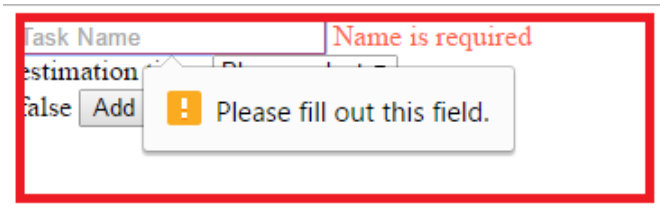

Figure 3.4: Example of an input field with a placeholder and a tooltip

3.5 Selenium

Selenium is a browser automation tool widely used for web testing that uses JavaScript to lodge the test automation motor into the browser.

The main functionality of this tool is the automatic control of the browser in order to systematize repetitive tasks, however there are other uses for it and as the official documentation of the tool states "*Selenium automates browsers. That's it! What you do with that power is entirely up to you*" [Sel18].

In the context of this solution, the browser automation functionality is also used, but the main purpose this tool is used for is for serving as an interface between the browser and the patterns being implemented, allowing to explore the elements of the web pages as well as gathering some information about them. In order to use this functionalities the WebDriver API is used.

3.6 Summary

In conclusion, the process of finding usability patterns was far more complex than what was expected, and ended up inducing further studies relating to usability principles and guidelines and how to adapt them into patterns.

The two usability areas chosen to be explored: Consistency and Error Prevention, ended up as promising paths to take since they account for a large part of the usability failures that occur, as well as being complex enough to offer the chance of finding multiple patterns related to them.

Concerning the patterns analyzed, three were found and chosen as good alternatives for the implementation, since they proved to be implementable, useful and representative of usability principles. Their implementation details will be covered in the next chapter.

Chapter 4

Implementation

This chapter will focus on providing a lower level description of the solutions explained in the previous chapter. The patterns implemented will be covered, as well as additional support functionalities.

4.1 GUI

In order to interact with the user, and to supply him with an easier way for him to provide the system the necessary inputs as well as the user configurations, a simple GUI was developed as shown in the images below.

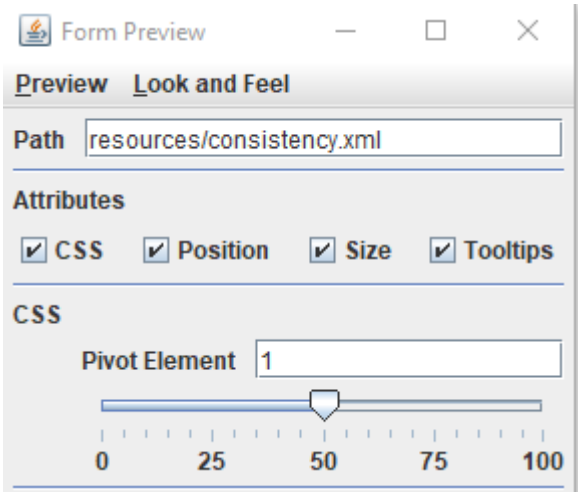

Figure 4.1: Application GUI - top part

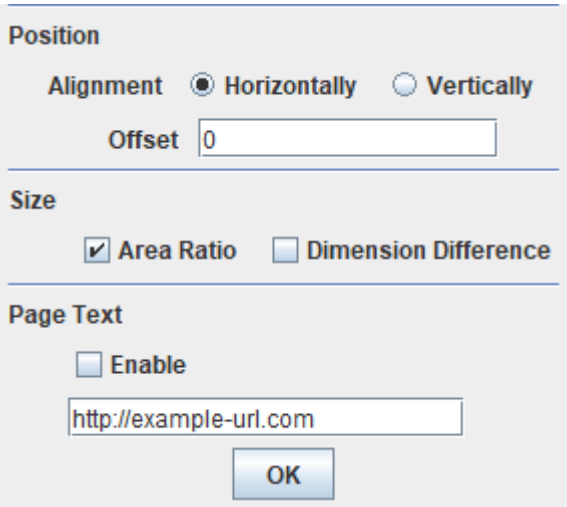

Figure 4.2: Application GUI - bottom part

The GUI exhibits several inputs, that account for variables that the user needs to fill in order to run the test patterns. The meaning of each of this inputs is:

- The **Path textbox** specifies the path to the XML file containing the user's input regarding the URL of the applications under test as well as the locators for the elements under test for the "Element Layout Consistency Pattern" and the "Tooltips and Placeholders Pattern". The content of this file is explained further in this chapter.
- The "**CSS**", "**Size**" and "**Position**" **checkboxes** act as enablers for the three test components applicable for the "Element Layout Consistency Pattern", then if we want to apply this pattern we should check these three checkboxes, if one of the components is not relevant for the test case, the respective checkbox can be left unchecked.
- The "**Tooltips**" **checkbox** is the enabler for the "Tooltips and Placeholders Pattern", leaving this checkbox checked will apply the pattern with the attribute values contained in the file defined in the Path field.
- The **Pivot Element** field indicates what is the core element with which the other elements should be compared to for layout consistency in the "Element Layout Consistency Pattern". The field should be provided with an integer corresponding to an element defined in the XML file.
- The slider is used to define the desired accuracy for the CSS style comparison, if the defined value is higher, more attributes should be matched between elements in order to be considered consistent.
- The "**Alignment**" radio buttons define whether the application should check the vertical or horizontal alignment between the elements, and the "**Offset**" field sets the maximum allowed distance (in pixels) the elements can be apart in order to still be considered aligned.

- The "**Area Ratio**" and "**Dimension difference**" checkboxes are relative to the "Size" component of the "Element Layout Consistency Pattern" and are only relevant if the "Size" checkbox previously mentioned is active. The checkboxes enable or disable the two types of analysis available for this component of the pattern.
- In the "**Page Text**" section are contained the inputs for the "Page Text Consistency Pattern". The checkbox simply enables/disables the execution of this pattern and the textbox requires the insertion of the URL of the page desired to be tested.

4.2 Input Files and Parsers

One of the main concerns of the implementation of this project was to build it in a versatile way that would support the integration with other existing tools or projects, such as the PBGT. As a result the way of inputing data to the system that will be described next was chosen, because it is easy to replicate if the inputs are gathered in any other way by the core tool, or easy to substitute by another input method.

In order to specify the URLs and element locators needed to execute some of the patterns a comprehensive XML format was defined as shown in the example below.

```
<consistency>
    <page>
        <url>http://example-url.net/</url>
        <element><![CDATA[/html/body/div[@id="first-element-byXpath"]]]></element>
        <name>second-element-name</name>
        <id>third-element-id</id>
        <class>fourth-element-class</class>
    </page>
    <page>
        <url>http://another-url.net/</url>
        <id>fifth-element-id</id>
        <id>sixth-element-id</id>
    </page>
</consistency>
```

Figure 4.3: An illustration of the XML format chosen

The file's composition consists in a root tag, named 'consistency' in this case, that may contain one or more '**page**' elements inside it. Only 'page' elements should be added to the root. Even though there can be as much 'page' instances as needed, a higher number will translate into a bigger time needed for data collection in the execution of the tool.

The 'page' tag contains several other tags, each with its own purpose, for each 'page' there should be:

- One, and only one, '**url**' tag containing the URL path to the desired page.
- One or more element locator tag, in their turn these tags can belong to one of four types:

- The type '**element**' is for locating the element based on a XPATH query. If this input method is adopted the user must make sure that the query provided is unique to the desired element, otherwise the selection may return another one. The XPATH query must be inside the <![CDATA[...]]> stub for parsing purposes.
- The '**name**' type searches the respective element by its 'name' HTML tag, therefore its content should correspond to this attribute value of the desired element.
- The '**id**' type content should be the same as the HTML id of the element on order to find it.
- Similarly the '**class**' element should correspond to the element's class.

The complete XML file contains the components to provide the system with the necessary inputs regarding the elements under test. Thus, after supplying the system with the file, and executing the tool, it will start by parsing the information contained in the file in order to gather the right information to find these pages and elements.

Since in the user configuration shown in the previous section the user must define the pivot element for the comparisons it is useful to have an idea of the parsing order of the elements' information contained in the XML file. The first order the parser follows is the pages in order of definition in the file, pages that are defined first are parsed first. For each page, the elements are parsed first according to the type of the tag provided in the following order: element>name>id>class, meaning that the all the elements located with an 'element' tag from a given page are parsed before all the 'id' defined elements of the same page. For elements of the same page, and with the same tag type the parsing order is once again the definition order. The elements of the figure 4.3 are named in the same order as they are parsed as an example.

In order to check if the data was parsed correctly and properly passed down to the patterns' execution and output is generated, below is the output generated for the figure 4.3 example.

```
Root element :consistency
Number of Pages: 2
---------------------------
Page URLS Parsed so far: [http://example-url.net/]
Last page xpath elements: [/html/body/div[@id="first-element-byXpath"]]
Last page name elements: [second-element-name]
Last page id elements: [third-element-id]
Last page class elements: [fourth-element-class]

Page URLS Parsed so far: [http://example-url.net/, http://another-url.net/]
Last page xpath elements: []
Last page name elements: []
Last page id elements: [fifth-element-id, sixth-element-id]
Last page class elements: []

All xpath elements: [[/html/body/div[@id="first-element-byXpath"]], []]
All name elements: [[second-element-name], []]
All id elements: [[third-element-id], [fifth-element-id, sixth-element-id]]
All class elements: [[fourth-element-class], []]
```

Figure 4.4: Output generated by the parser

4.3 Patterns

4.3.1 Element Layout Consistency Pattern

The execution of this pattern is composed of the following stages:

1. In the first place, a file containing a list of CSS style attributes relevant to the analysis is loaded to memory in order to make the access to this information faster, the file containing the attributes can be changed, if for any reason there is the need to add more CSS attributes to the list or remove some that are not useful.

2. Using the information parsed from the XML file, the pattern execution and the web pages defined will interact with Selenium tool acting as an interface, the procedure of this interaction for each page consists in:

   - Establish a connection with the web page using Selenium

```
1 //testUrls represents an ArrayList with the URLs parsed from the XML file
2
3 DriverHandler.getDriver().get(testUrls.get(i));
```

   - For every type of element locators (xpath, name, id and class) go through a list of the parsed elements of this page and proceed to find them.

```
1 for(int j=0; j<testXpaths.get(i).size(); j++) {
2
3   WebElement elem = DriverHandler.getDriver().findElementByXPath(
        testXpaths.get(i).get(j));
4 }
```

```
1 for(int j=0; j<testNames.get(i).size(); j++) {
2
3   WebElement elem = DriverHandler.getDriver().findElementByName(testNames
        .get(i).get(j));
4 }
```

```
1 for(int j=0; j<testIds.get(i).size(); j++) {
2
3   WebElement elem = DriverHandler.getDriver().findElementById(testIds.get
        (i).get(j));
4 }
```

```
for(int j=0; j<testClasses.get(i).size(); j++) {

  WebElement elem = DriverHandler.getDriver().findElementByClassName(
      testClasses.get(i).get(j));
}
```

- For each of the found elements in the page store the information to be analyzed afterwards.

```
//CSS information
allElemsCSS.add(getElementCSSValues(attributes, elem));
//Size information
sizes.add(elem.getSize());
//Position information
locations.add(elem.getLocation());
```

3. After establishing the connection with all the pages and gathering information about the elements each component's information is parametrized and analyzed in order to compile the outputs to be presented to the user.

```
    if(css) {
            System.out.println("CSS: ");
            runTestCss(pivot, cssPercentage, cssattr);
            System.out.println();
        }
        if(position) {
            System.out.println("Position: ");
            runTestPosition(pivot, horizontalAlignment, positionOffset);
            System.out.println();
        }
        if(size) {
            System.out.println("Size: ");
            runTestSize(pivot, areaRatio, dimensionDiff);
        }
```

4. Finally, the output is displayed to the user.

- CSS output:

```
CSS:
Pivot(2) and Element 1
9/115 Css values are different

color
column-rule-color
font-family
font-size
height
outline-color
perspective-origin
text-decoration-color
width
```

This is a case in which the comparison between the selected pivot element and one of the others elements gave a negative result, therefore the style attributes that are different are also showed.

```
Pivot(2) and Element 3
2/115 Css values are different

CSS is similar.
```

In this case the comparison between the elements is sufficiently similar according to the user defined standards.

- Alignment output:

```
Position:
Elements 2 and 1 are NOT aligned 63 units apart

Elements 2 and 3 are aligned
```

The first case represents a failed test scenario, while the second one is a successful one.

- Size output:

```
Size:
Area ratio between elements 2 and 1 is 0.7601351
Height difference between elements 2 and 1 is -3
Width difference between elements 2 and 1 is -11

Area ratio between elements 2 and 3 is 1.016129
Height difference between elements 2 and 3 is 0
Width difference between elements 2 and 3 is 1
```

4.3.2 Page Text Consistency Pattern

The stages that make up this pattern are:

1. Loading CSS attributes that relate with text formatting and establishing a connection with the page defined by the user in the configuration using Selenium

2. Explore the page to find and process every single element contained in it. For every element this processing consists in:

   - Check if the element contains a text field.
   - Execute an algorithm to make sure the text field belongs to the element itself and not to one of its children elements.
   - If the steps above don't fail, a reference to the element is saved for its future analysis.

```
for(int i=0; i< allElements.size(); i++){

  WebElement e = allElements.get(i);

  String text = e.getText();
  for (WebElement child : e.findElements(By.xpath("./*"))) {
    text = text.replaceFirst(child.getText(), "");
  }

  if(!text.trim().equals("")){
    indexList.add(i);
  }
}
```

3. For every element stored as a result of the previous selection its text formatting will be extracted and compared to the ones already checked to verify if its unique or the same as another one.

```
for(int i : indexList){

  List<String> elemTextCss =getElementCSSValues(textCss, allElements.get(i));

  if(cssValues.contains(elemTextCss)){

    for(int j=0; j < cssValues.size(); j++){
      if(cssValues.get(j).equals(elemTextCss)){
      cssIndex.add(j);
      }
    }

  }else {
    cssValues.add(elemTextCss);
    cssIndex.add(cssValues.size() -1);
  }
}
```

4. Lastly, the elements are then grouped with other ones formatted in the same way. These groups are then showed to the user in the format shown in 4.1.

```

Values 7
Boas vindas
Orgaos de Gestao
Departamentos
Servicos
Estudantes
Pessoal
Cursos
I&D e Inovacao
Cooperacao
Candidatos
Alumni
Empresas
Noticias
Pesquisa

color: rgba(140, 45, 25, 1)
direction: ltr
letter-spacing: normal
line-height: 17.3714px
```

```
22 text-decoration-color: rgb(140, 45, 25)
23 text-decoration-line: none
24 text-decoration-style: solid
25 text-indent: 0px
26 text-shadow: none
27 text-transform: none
28 white-space: normal
29 word-spacing: 0px
```

Listing 4.1: Example of a text formatting group output

4.3.3 Tooltips and Placeholders Pattern

The first stage of the execution of this pattern is similar to the second stage of the "Element Layout Consistency Pattern", since the input with the information of the elements to be tested is the same, the parsing is done in the same way as well as the access to the pages and the respective elements. However when the elements are scanned for information its where the patterns start to diverge.

Accordingly, in that phase of execution this pattern proceeds by checking each element if they belong to the 'input' format (the type of element relevant to be tested by this pattern), if the elements confirm to be testable inputs the algorithms for finding tooltips and placeholders are executed.

```
1  Actions builder = new Actions(DriverHandler.getDriver());
2
3  tooltip = elem.getAttribute("title");
4
5  if (tooltip.equals(""))
6  {
7
8    Action mouseOver = builder.moveToElement(elem).build();
9    mouseOver.perform();
10
11   try {
12   tooltip = DriverHandler.getDriver().findElementById("tooltip").getText();
13   } catch (NoSuchElementException e) {
14   }
15 (...)
```

Listing 4.2: Excerpt of the tooltip finding algorithm

After the elements are analyzed for tooltips and placeholders an output is generated for the user telling if these aspects were found and what are the respective values.

```
Tooltips:
Element not analyzed for tooltips
Tooltip not found
Tooltip value: Introduza o utilizador

Placeholders:
Element not analyzed for placeholders
Placeholder not found
Placeholder value: Utilizador
```

Listing 4.3: Output generated by the Tooltips and Placeholders Pattern

In the example output in 4.3 we can see the three possible scenarios regarding the analysis of an element.

- The first element is not analyzed because it is not a valid 'input' type
- The second element either doesn't have a tooltip and a placeholder or they are inaccessible by the tool
- The last element has a tooltip and a placeholder, therefore the tool presents their value to the user.

4.4 Summary

In this chapter, a lower level analysis of the implemented patterns was performed, in order to better illustrate how the recurring testing solutions for common usability issues were achieved. The objectives were matched since the patterns were able to be implemented, are able to be executed in a wide range of systems, test relevant aspects of the usability of web systems and are suited to be integrated with other previously implemented systems.

The next chapter will explore the behavior of the patterns in case study experiments in order to assess and validate the approach chosen.

Chapter 5

Experiments

This chapter will present case studies that will consist in executing the application's patterns in real web systems analyzing the results.

Operating System	Windows 10 Home
CPU	Intel Core i7-4700MQ 2.4GHz
Memory	8 GB

Table 5.1: System Specifications

5.1 Element Layout Consistency Pattern

In the next section, the case study analysis for the Element Layout Consistency Pattern is presented.

5.1.1 Case Study - SiFEUP

For this case study of this pattern the web administration system from FEUP (SiFEUP) was chosen. Three different pages from the system were selected in order to check the layout consistency of some their elements. The pages in question are shown in 5.1, 5.2 and 5.3 as well as the elements under test that are highlighted in red. These elements have similar functions and should be consistent between them.

Figure 5.1: SiFEUP case study - first page

Figure 5.2: SiFEUP case study - second page

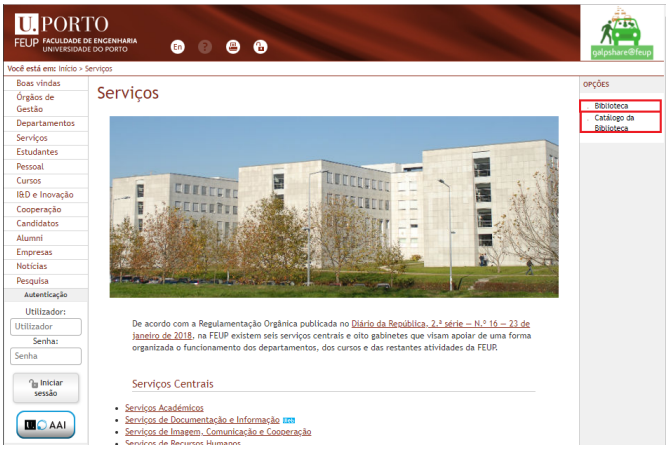

Figure 5.3: SiFEUP case study - third page

For the configuration of the test scenario described, the following XML input was used:

```
<consistency>
    <page>
        <url>https://sigarra.up.pt/feup/pt/web_base.gera_pagina?P_pagina=1182</url>
        <element><![CDATA[//*[@id="colunaextra"]/div[1]/div[2]/ul[1]/li[1]/a]]></
            element>
        <element><![CDATA[//*[@id="colunaextra"]/div[1]/div[2]/ul[1]/li[3]/a]]></
            element>
    </page>
    <page>
        <url>https://sigarra.up.pt/feup/pt/web_base.gera_pagina?p_pagina=259429</
            url>
        <element><![CDATA[//*[@id="colunaextra"]/div[1]/div[2]/ul/li[1]/a]]></
            element>
        <element><![CDATA[//*[@id="colunaextra"]/div[1]/div[2]/ul/li[3]/a]]></
            element>
    </page>
    <page>
        <url>https://sigarra.up.pt/feup/pt/uni_geral.nivel_list?pv_nivel_id=4</url>
        <element><![CDATA[//*[@id="colunaextra"]/div/div[2]/ul/li[1]/a]]></element>
        <element><![CDATA[//*[@id="colunaextra"]/div/div[2]/ul/li[2]/a]]></element>
    </page>
</consistency>
```

With the next user configurations:

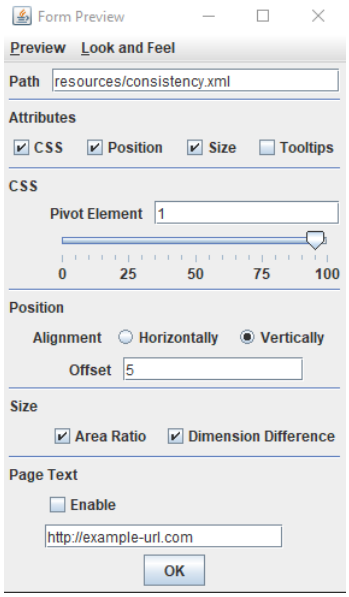

Figure 5.4: SiFEUP case study - user configurations

In this case, after a brief visual analysis of the elements, it was concluded that the results should show the elements to be fairly consistent in terms of style and aligned vertically. As seen in 5.4 the style sensitivity was set to 95% and the alignment offset to 5 pixels. The outputs after the execution were as follows:

```
CSS:
Pivot(1) and Element 2
0/115 Css values are different
CSS is similar.

Pivot(1) and Element 3
0/115 Css values are different
CSS is similar.

Pivot(1) and Element 4
0/115 Css values are different
CSS is similar.

Pivot(1) and Element 5
0/115 Css values are different
CSS is similar.

Pivot(1) and Element 6
0/115 Css values are different
CSS is similar.
```

Regarding the CSS analysis the results were as expected showing that the elements are all formatted in the same way revealing good consistency between them.

```
Position:
Elements 1 and 2 are aligned

Elements 1 and 3 are aligned

Elements 1 and 4 are aligned

Elements 1 and 5 are aligned

Elements 1 and 6 are aligned
```

In terms of Alignment the output shows that with the selected offset all the elements are vertically aligned with the pivot element, meaning that they are all within 5 pixels away horizontally in the page from the position of that pivot.

```
Size:
Area ratio between elements 1 and 2 is 0.48192772
Height difference between elements 1 and 2 is 0
Width difference between elements 1 and 2 is -43

Area ratio between elements 1 and 3 is 0.8
Height difference between elements 1 and 3 is 0
Width difference between elements 1 and 3 is -10

Area ratio between elements 1 and 4 is 0.43010753
Height difference between elements 1 and 4 is 0
Width difference between elements 1 and 4 is -53

Area ratio between elements 1 and 5 is 0.8333333
Height difference between elements 1 and 5 is 0
Width difference between elements 1 and 5 is -8

Area ratio between elements 1 and 6 is 0.31189084
Height difference between elements 1 and 6 is -15
Width difference between elements 1 and 6 is -17
```

The size analysis shows some differences between the elements' areas and dimensions, and this could potentially reveal a consistency flaw, however when analyzing this particular case and since the elements are text based it is normal, according to their text content, that their size can vary, therefore not meaning necessarily that the consistency flaw exists.

This case study showed that although some conclusions can be taken from automatic usability analysis, because usability itself is a user based concept, there should always be a critical analysis performed by an evaluator in order to prevent misunderstandings of the outputs.

5.2 Page Text Consistency Pattern

In the following section, it is introduced the case study analysis for the Page Text Consistency Pattern.

5.2.1 Case Study - Google

For this case study the Google home page was chosen as a test subject because it is a page with few text elements which simplifies the comprehension of the case, as well as distinct elements that are easy to analyze manually, therefore giving us means to evaluate the solution.

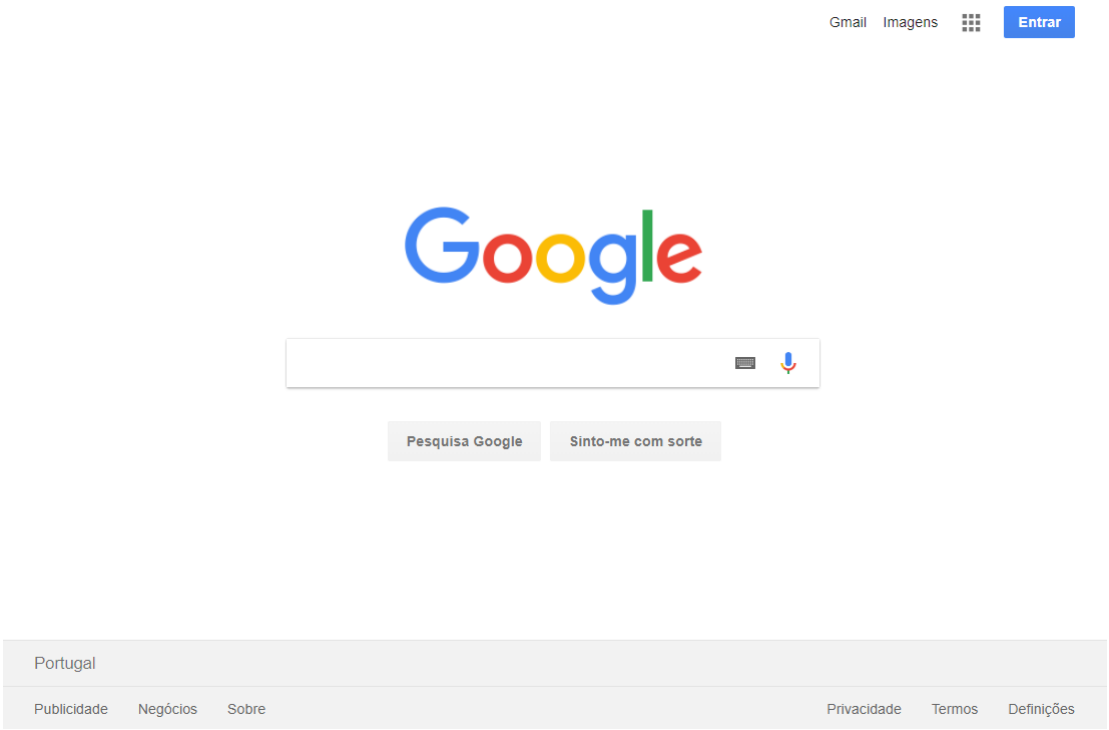

Figure 5.5: Google case study - Page

In order to run the pattern in the presented page the following configuration was provided to the system in the 'Page Text' section of the GUI:

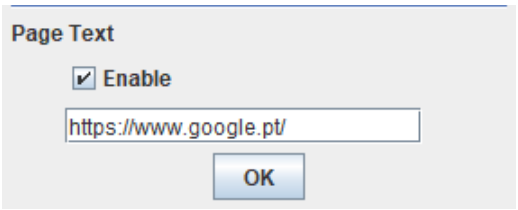

Figure 5.6: Google case study - Configuration

After the execution of the pattern in the given page, the following output was generated[1]:

```
1
2  Values 0
3  Gmail
4  Imagens
5
6  ____________________
7
8  Values 1
9  Entrar
10
11 ____________________
12
13 Values 2
14 Um lembrete de privacidade da Google
15
16 ____________________
17
18 Values 3
19 LEMBRAR-ME DEPOIS
20
21 ____________________
22
23 Values 4
24 REVER AGORA
25
26 ____________________
27
28 Values 5
29 Portugal
30
31 ____________________
32
33 Values 6
34 Privacidade
35 Termos
36 Definicoes
37 Publicidade
38 Negocios
39 Sobre
```

As can be seen from the output generated and similarly to the manual analysis performed beforehand, the text elements with the same relevance and/or purpose are grouped together if analyzed for their style layout. With this pattern it easier to detect the presence of consistency, as seen in this case, than the absence of it, since there could be a reason for values seeming unexpected in the output according to a previous visual analysis, thus when faced with the occurrence of an apparently bad text consistency evaluation from the output, the evaluators should have extra concern when analyzing the results.

[1]Note: The text CSS values from each of the element groups was omitted, for the complete output see the Appendix

5.3 Tooltips and Placeholders Pattern

The next section will cover the analysis of the case study regarding the error prevention 'Tooltips and Placeholders Pattern'.

5.3.1 Case Study - UPORTO

For this case study the authentication page of the UPORTO website was chosen, due to the fact that it contains several types of elements in the same page that should all be dealt with in different ways by the system.

Below, in 5.7, is showed the authentication page of the UPORTO website, with the elements chosen to be tested highlighted in red and numbered according to their processing order.

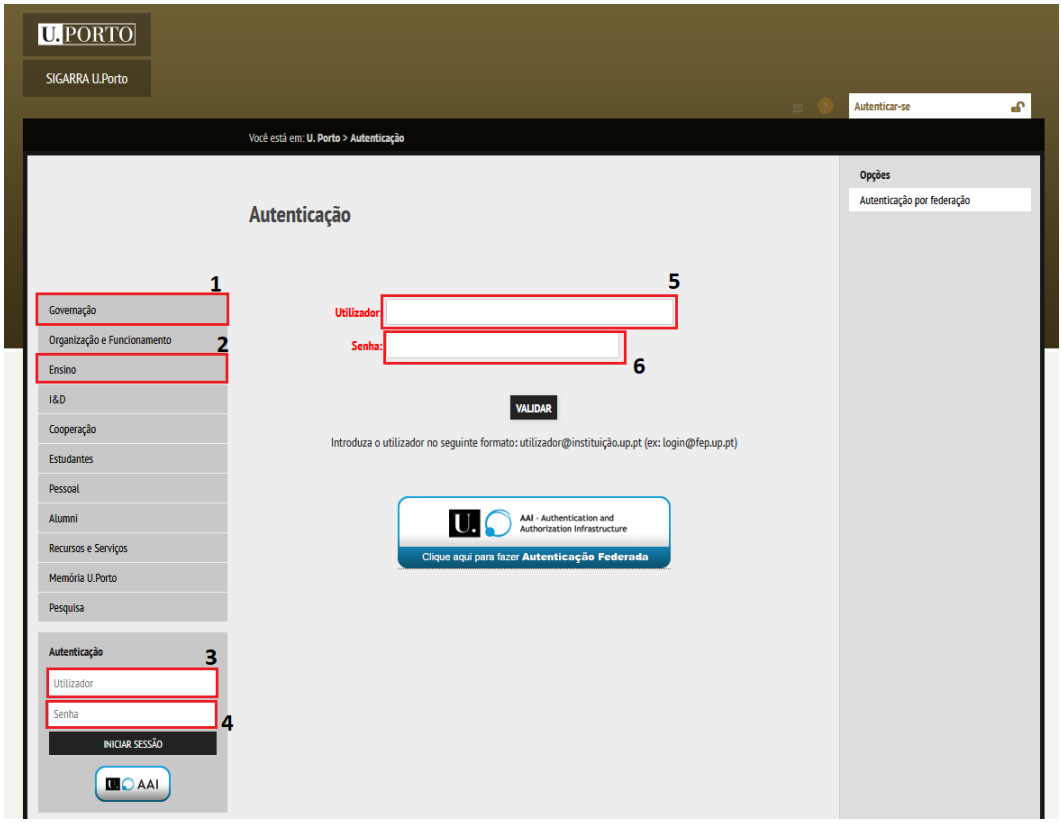

Figure 5.7: Uporto case study - Authentication Page

The XML input file was defined as:

```
1 <errprev>
2     <page>
3         <url>https://sigarra.up.pt/up/pt/vld_validacao.login?p_address=WEB_PAGE.
              INICIAL&amp;p_app=891&amp;p_amo=55</url>
4         <element><![CDATA[//*[@id="menu-navegacao-conteudo"]/ul/li[1]/a]]></element
              >
```

```
5         <element><![CDATA[//*[@id="menu-navegacao-conteudo"]/ul/li[3]/a]]></element
              >
6         <id>user</id>
7         <id>pass</id>
8         <id>p_user</id>
9         <id>p_pass</id>
10    </page>
11 </errprev>
```

The execution of the pattern with these inputs resulted in the following output:

```
1  Tooltips:
2  Element not analyzed for tooltips
3  Element not analyzed for tooltips
4  Tooltip value: Introduza o utilizador no seguinte formato: utilizador@instituicao.
       up.pt (ex: login@fep.up.pt)
5  Tooltip value: Introduza a senha
6  Tooltip not found
7  Tooltip not found
8
9  Placeholders:
10 Element not analyzed for placeholders
11 Element not analyzed for placeholders
12 Placeholder value: Utilizador
13 Placeholder value: Senha
14 Placeholder not found
15 Placeholder not found
```

The first two elements were not searched for tootlips and placeholders since their type is not a valid one to be analyzed by this pattern.

The next two elements were analyzed and their tooltip an placeholder values were presented to the user in the output.

For the last two elements, although they were from a valid type, the execution didn't find a tooltip nor a placeholder. This could mean one of two things, either the element didn't have a tooltip and a placeholder, or the pattern wasn't able to find them despite their existence. After a manual analysis to the elements it is possible to verify that the case was that the element didn't possess these attributes.

5.4 Summary

The case studies presented in this chapter allowed to better understand the functionalities implemented as well as some of their limitations. The test patterns proved to be generic enough to test the recurring issues in different web systems along with being a useful tool to better understand the usability of the systems. However, it was also verified that the patterns alone still don't have

the ability to reach reliable conclusions regarding the usability of the system. The intervention of evaluators is still needed to further explore usability issues that don't have the possibility of being totally automatized and to assess the outputs provided by the patterns since sometimes the conclusions to be taken from them might not be obvious.

Even though this approach is not a standalone solution to test usability it still automates some usability aspects making the test process more efficient, and the usability evaluations more complete and thorough.

Chapter 6

Conclusions and Future Work

In this last chapter, the work done in this Dissertation is wrapped up, also exposing possible aims for future work related to this project.

6.1 Summary

This dissertation had the objective of studying, finding and implementing usability test patterns, or in other words generic test solutions for recurring usability problems, in order to introduce some automation in the usability testing processes that are typically very human based.

In order to explore usability patterns two branches of the usability field were examined: consistency and error prevention. As a result three patterns were deduced:

- **Element Layout Consistency Pattern**: Testing consistency between elements with the same purpose or function in a web system.

- **Page Text Consistency Pattern**: Testing the similarity between text with the same relevance in a page.

- **Tooltips and Placeholders Pattern**: Testing the presence of tooltips and placeholders that enhance error prevention in input elements.

These patterns were successfully implemented and proved to be plausible generalizations of usability issues. Although they are not able to act alone as a usability evaluation without the influence of human evaluators, it certainly is a valuable asset to accompany the existing testing methods also introducing a new degree of automation in those same processes.

6.2 Future Work

In terms of future work related to this Dissertation, there are three main paths that can be looked into:

- Finding and implementing new usability patters. The slice of usability explored in the context of this project is very thin, therefore there is still plenty of room to find recurring behaviors and consequently new patterns.
- Improving the implemented patterns. Despite the patterns fulfilling the objectives proposed, they are still not completely optimized and there are aspects that if looked into in further detail would improve the performance and efficiency of this tool, for instance, the time efficiency of the page exploration algorithm is acceptable when we have a small number of pages and elements, however if a very large sample of pages were provided, the runtime of the application would also be very high, another example is the accuracy when finding tooltips, since the tooltips can be created dynamically in real time using technologies such as JavaScript, in some of these cases the pattern is not able to pick up existing tooltips providing the user with false negative results.
- Finally, there is the option of extending existing testing suites such as the PBGT to support the integration of usability patterns, as stated before the implementation of the patterns was built so that this process should be easier.

Despite its limitations, this project allowed to conclude that the array of options for automating and generalizing usability is wider than what was expected, making way for future projects that can explore this field using similar approaches.

References

[AP18] Paulo J. M. Araújo and Ana C. R. Paiva. Pattern-based web security testing. In *6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD*, 2018.

[AS07] Richard Atterer and Albrecht Schmidt. Tracking the interaction of users with AJAX applications for usability testing. *Proceedings of the 25th SIGCHI Conference on Human Factors in Computing Systems*, pages 1347–1350, 2007.

[Cho12] Paras Chopra. The Ultimate Guide To A/B Testing. Usability Testing Central, available at http://www.smashingmagazine.com/2010/06/24/the-ultimate-guide-to-a-b-testing/, 2012.

[CPFA10] Marco Cunha, Ana C. R. Paiva, Hugo Sereno Ferreira, and Rui Abreu. Pettool: A pattern-based gui testing tool. In *2nd International Conference on Software Technology and Engineering (ICSTE)*, volume 1, pages 202–206, 2010.

[Cra13] CrazyEgg. CrazyEgg - Visualize where your visitors click, 2013.

[Dic02] R Stanley Dicks. Mis-usability : On the uses and misuses of usability testing. *SIGDOC '02 Proceedings of the 20th annual international conference on Computer documentation*, pages 26–30, 2002.

[DP17] F. Dias and A. C. R. Paiva. Pattern-based usability testing. In *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 366–371, March 2017.

[Dum03] Joseph S. Dumas. The human-computer interaction handbook. chapter User-based Evaluations, pages 1093–1117. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 2003.

[Fes17] Therese Fessenden. Horizontal Attention Leans Left. Usability Testing Central, available at https://www.nngroup.com/articles/horizontal-attention-leans-left/, 2017.

[JHJ98] Niels Ebbe Jacobsen, Morten Hertzum, and Bonnie E. John. The evaluator effect in usability tests. *Chi 1998*, (APRIL):255–256, 1998.

[KCF92] Claire-Marie Karat, Robert Campbell, and Tarra Fiegel. Comparison of empirical testing and walkthrough methods in user interface evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, pages 397–404, 1992.

REFERENCES

[LPWR90] Clayton Lewis, Peter G. Polson, Cathleen Wharton, and John Rieman. Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems Empowering people - CHI '90*, pages 235–242, 1990.

[MD97] Gerard Meszaros and Jim Doble. Pattern languages of program design 3. chapter A Pattern Language for Pattern Writing, pages 529–574. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[Mem02] Atif M. Memon. Gui testing: Pitfalls and process. *Computer*, 35(8):87–88, August 2002.

[MP14a] M. L. M. Rodrigo Moreira and Ana C. R. Paiva. Towards a Pattern Language for Model-Based GUI Testing. *19th European Conference on Pattern Languages of Programs (EuroPLoP 2014)*, 2014.

[MP14b] Rodrigo Moreira and Ana Paiva. A gui modeling dsl for pattern-based gui testing - paradigm. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'14)*, 2014.

[MP14c] Rodrigo Moreira and Ana C. R. Paiva. Towards a pattern language for model-based gui testing. In *19th European Conference on Pattern Languages of Programs*, 2014.

[MP14d] Rodrigo M.L.M. Moreira and Ana C.R. Paiva. PBGT Tool: An Integrated Modeling and Testing Environment for Pattern-based GUI Testing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 863–866, 2014.

[MPM13] Rodrigo M L M Moreira, Ana C R Paiva, and Atif Memon. A Pattern-Based Approach for GUI Modeling and Testing. In *24th IEEE International Symposium on Software Reliability Engineering (ISSRE2013)*, Pasadena, CA, USA, November 2013.

[MPNM17] Rodrigo Moreira, Ana Paiva, Miguel Nabuco, and Atif Memon. Pattern-based GUI testing: Bridging the gap between design and quality assurance. In *Journal of Software: Testing, Verification and Reliability (STVR)*, volume 27, March 2017.

[Nie88] Jakob Nielsen. Coordinating user interfaces for consistency. 20(3):15–16, 1988.

[Nie92] Jakob Nielsen. Finding usability problems through heuristic evaluation. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '92*, pages 373–380, 1992.

[Nie94] Jakob Nielsen. Usability inspection methods. *Conference companion on Human factors in computing systems - CHI '94*, pages 413–414, 1994.

[Nie05] Jakob Nielsen. Ten Usability Heuristics. *Communications of the ACM*, 3(1990):1–2, 2005.

[PN09] Kara Pernice and Jakob Nielsen. Eyetracking Methodology. *... and Evaluate Usability Studies Using Eyetracking. ...*, (August):159, 2009.

[PV17] Ana C. R. Paiva and Liliana Vilela. Paradigm-cov tool. In *Cluster Computing Journal*, 2017.

REFERENCES

[RCC+02] Stephanie Rosenbaum, Gilbert Cockton, Kara Coyne, Michael Muller, and Thyra Rauch. Focus groups in hci: Wealth of information or waste of resources? In *CHI '02 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '02, pages 702–703, New York, NY, USA, 2002. ACM.

[RRH00] Stephanie Rosenbaum, Janice Anne Rohn, and Judee Humburg. A toolkit for strategic usability. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '00*, pages 337–344, 2000.

[Sav96] Pamela Savage. User interface evaluation in an iterative design process: a comparison of three techniques. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 307–308, 1996.

[SB97] Elizabeth J. Simeral and Russell J. Branaghan. Comparative analysis of heuristic and usability evaluation methods. In *Proceedings/STC, Society for Technical Communication Annual Conference*, pages 307–309, 1997.

[Sel18] Selenium. SeleniumHQ - Browser automation. Usability Testing Central, available at https://www.seleniumhq.org/, 2018.

[Shn10] Ben Shneiderman. The Eight Golden Rules of Interface Design. In *Shneiderman, B. and Plaisant, C., Designing the User Interface: Strategies for Effective Human-Computer Interaction: Fifth Edition*, page 606. 2010.

[SP14] Clara Sacramento and Ana C. R. Paiva. Web application model generation through reverse engineering and ui pattern inferring. In *9th International Conference on the Quality of Information and Communications Technology (QUATIC'14)*, 2014.

[Spi07] Frank Spillers. Formal vs. informal usability tests. Usability Testing Central, available at http://www.usabilitytestingcentral.com/2007/04/formal_vs_infor.html, April 2007.

[Wan16] Yuan Wang. Ux research communication – informal usability tests. Interaction Design Foundation, available at https://www.interaction-design.org/literature/article/ux-research-communication-informal-usability-tests, 2016.

REFERENCES

Appendix A

A.1 Google Case Study Complete Output

```
Values 0
Gmail
Imagens

color: rgba(0, 0, 0, 0.87)
direction: ltr
letter-spacing: normal
line-height: 24px
text-decoration-color: rgba(0, 0, 0, 0.87)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: none
white-space: nowrap
word-spacing: 0px
----------------------

Values 1
Entrar

color: rgba(255, 255, 255, 1)
direction: ltr
letter-spacing: normal
line-height: 28px
text-decoration-color: rgb(255, 255, 255)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: none
white-space: nowrap
word-spacing: 0px
```

```
----------------------

Values 2
Um lembrete de privacidade da Google

color: rgba(0, 0, 0, 0.87)
direction: ltr
letter-spacing: normal
line-height: 40px
text-decoration-color: rgba(0, 0, 0, 0.87)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: none
white-space: normal
word-spacing: 0px
----------------------

Values 3
LEMBRAR-ME DEPOIS

color: rgba(66, 133, 244, 1)
direction: ltr
letter-spacing: normal
line-height: 16px
text-decoration-color: rgb(66, 133, 244)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: uppercase
white-space: nowrap
word-spacing: 0px
----------------------

Values 4
REVER AGORA

color: rgba(255, 255, 255, 1)
direction: ltr
letter-spacing: normal
line-height: 16px
text-decoration-color: rgb(255, 255, 255)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: uppercase
```

```
white-space: nowrap
word-spacing: 0px
----------------------

Values 5
Portugal

color: rgba(0, 0, 0, 0.54)
direction: ltr
letter-spacing: normal
line-height: 40px
text-decoration-color: rgba(0, 0, 0, 0.54)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: none
white-space: normal
word-spacing: 0px
----------------------

Values 6
Privacidade
Termos
Definicoes
Publicidade
Negocios
Sobre

color: rgba(102, 102, 102, 1)
direction: ltr
letter-spacing: normal
line-height: 40px
text-decoration-color: rgb(102, 102, 102)
text-decoration-line: none
text-decoration-style: solid
text-indent: 0px
text-shadow: none
text-transform: none
white-space: nowrap
word-spacing: 0px
----------------------
```