EIM Faculty

Human-Computer Interaction Research Group

Bachelor Thesis

Concept and Implementation of a Smart Web-Application for Heuristic Evaluation Based on Data Mining Methods

By: Isaac Rahnema

Supervisors:

Prof. Dr. Gerd Szwillus

Prof. Dr. -Ing Reinhard Keil

Bachelor Thesis

Department of Computer Science
Faculty for Electrical Engineering, Computer Science and Mathematics
Paderborn University

By:
Isaac Rahnema
erahnema@mail.uni-paderborn.de
Student No. 6751668

Supervised by:
Prof. Dr. Gerd Szwillus
Prof. Dr. -Ing Reinhard Keil

Submitted to the Human-Computer Interaction Research Group
In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

Erklärung der Urheberschaft

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum Unterschrift

Table of Content:

1 Introduction

The process of systematic development of user friendly interfaces or *usability engineering* involves constant evaluation of a system design through *usability tests* and/or *inspection methods*. *Heuristic Evaluation* (*HE*) is one of the most important inspection methods in usability engineering, in which a number of usability experts evaluate the user interfaces based on a set of pre-defined principles or the so called *heuristics* and at the end, their findings will be merged in a summary report.

This work is about the design and implementation of *SHE* or *Smart Heuristic Evaluation* (*www.smarthe.net*) as a collaborative web application for supporting heuristic evaluation. The concept of usability engineering is defined and explained in chapter 2 to see what the role of inspection methods like HE is in the usability engineering process. In the third chapter, conducting heuristic evaluation is discussed to see which part of it could be supported by tools. The existing tools for HE are discussed in chapter 4 to see which shortcomings of them could/should be improved. Chapters 5-6 discuss how SHE is designed based on MVC architecture (see chapter 5) and which technologies are used in the implementing of SHE (see chapter 6). Some selected implementation aspects of SHE is described in chapter 7 and finally, chapter 8 presents a walkthrough of SHE to see how it can be used for HE. Appendix A gives the necessary steps for running SHE in a local machine using the source code which can be accessed in the following *git* repository: *https://erahnema@git.cs.upb.de/erahnema/she.git*.[1] In the appendix B the result of a HE test project is represented which is conducted using SHE.

[1] Alternatively the source code can be accessed in *github*: *https://github.com/boogh/she.git*

2 Usability Engineering

This chapter introduces the concept of *Usability Engineering (UE).* To understand UE, first we should understand the concept of *usability*.

2.1 Usability

The term *usability* is an attribute of a system which is defined, according to ISO 9241 part 11 (ISO 9241-11, 1998), as "the extent to which a system, product or service can be used by specified users to achieve specified goals with *effectiveness*, *efficiency* and *satisfaction* in a specified context of use". *Effectiveness* emphasize the "accuracy and completeness with which users achieve specified goals", in another word, how well users can accomplish their task using a specific system interface. *Efficiency* refers to the "resources used in relation to the results achieved", resources like time, human effort, costs and materials. And finally, *satisfaction* describes "person's perceptions and responses that result from the use of a system, product or service", or in simple words, how the user feels during and after using a system.

Nielsen (Nielsen J. , Usability Engineering, 1993) sees usability as the part of the more general concept *system acceptability* (see image 2.1), or "the question of whether the system is good enough to satisfy all the needs and requirements of users" (Nielsen J. , Usability Engineering, 1993, p. 24). System acceptability could refer to the social and practical acceptability. A system can be practically acceptable but socially rejected. To be practically accepted, a system should be acceptable in various fields, e.g. cost, support, reliability, compatibility with other existing systems etc. A system also should be *useful*. *Usefulness* refers to the question whether a system can be used to achieve desired goals. Usefulness breaks down[2] to the two categories: *utility*, i.e. "the functionality of a system to achieve a desired goal", and *usability*, i.e. "how well a user can use that functionality". Therefore usability includes all the aspects of a system with which a human might interact.

[2] For these two categories Nielsen refers to: Grudin, J. (1992). Utility and usability: Research issues and development contexts. *Interacting with Computers* 4, 2 (August), 209-217.

Nielsen sees usability as a multi-dimensional property of a user interface which can be associated with five attributes (Nielsen J. , Usability Engineering, 1993, p. 25):

1) Learnability: The system should be easy to be learned

2) Efficiency: After learning, working with the system should be productive.

3) Memorability: No need to learn twice, which means it should be easy to remember how to use a system.

4) Errors: Low error rate with easy recovering.

5) Satisfaction: User should like to work with the system or subjective satisfaction is the result of working with the system.

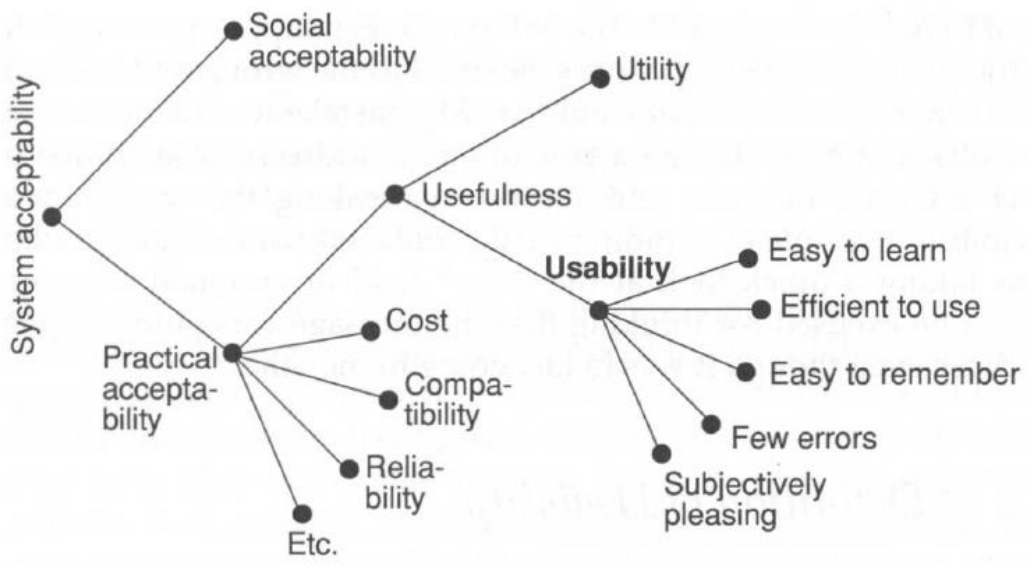


Image 2.1: Usability as a part of a more general concept (Nielsen, 1993, p25)

2.2 Usability Engineering (UE)

Usability Engineering (UE) refers to the methods of systematic development of the user friendly interfaces. This process of user centered design (UCD) involves, as Rubin (Rubin, 1984) emphasizes, early focus on the users and tasks, empirical measurement and testing of product usage through prototype testing with actual users and finally *iterative design*, which means the product should be designed and tested repeatedly. This process of iterative improvement starts from the beginning of the development process and continues throughout the whole production process. ISO 13407 (ISO 13407, 1999) gives general guidelines about this iterative process which can be seen in image 2.2.

The iterative circle of improving of UI, as ISO stated, involves four stages:

1) First step is the understanding and specifying the context of use. *Context of use* refers to the questions like: Who are the future users and what are their characteristics (e.g. user skills, education, preference, capabilities)? What kind of tasks these users want to performs using this system? And finally: In what kind of environment the system will be deployed (e.g. hardware, software)?
2) In the second step user and organizational requirements are concretely defined. These requirements can include operational and financial conditions, legislative terms (e.g. safety and health), cooperation and communication between users and other involved parties, conditions of operation and maintenance, task performance etc. This stage should end up in defining concrete usability criteria, upon which a set of defined targets (based on tradeoffs between different requirements) could be tested.
3) Third step involves producing design solution based on the state of art experiences, knowledge and the result of the analyses of the context of use. This design can involve simulation, modeling and prototyping and also redesigning based on the feedbacks from user tests.
4) Evaluation against requirements is a crucial step in each cycle, which can be done using experts, users or both of them. Bringing new and deeper understandings of the context of using and working as a base to asses if the user and organizational objectives are fulfilled, iterative evaluation gives valuable feedbacks in early stages of development, that helps to improve a design which makes the product more economic both in time and financially. Early feedbacks should not necessarily be based on user experiences, which only can be possible in the later stages of development with a runnable prototype, but can be obtained from expert reviews. Expert evaluations can be done in any stage of development and it is fast and certainly cheaper than a user test, because it can happen in early stages of design and most major usability issues can be fixed with redesigning, which is much cheaper that changing the end product. Evaluation should be continued by monitoring the product in a long-term period because some aspects of the system are only conceivable in a longer period of time and this makes it more clear why usability engineering is a process which is interwoven with all the stages of software development and maintenance.

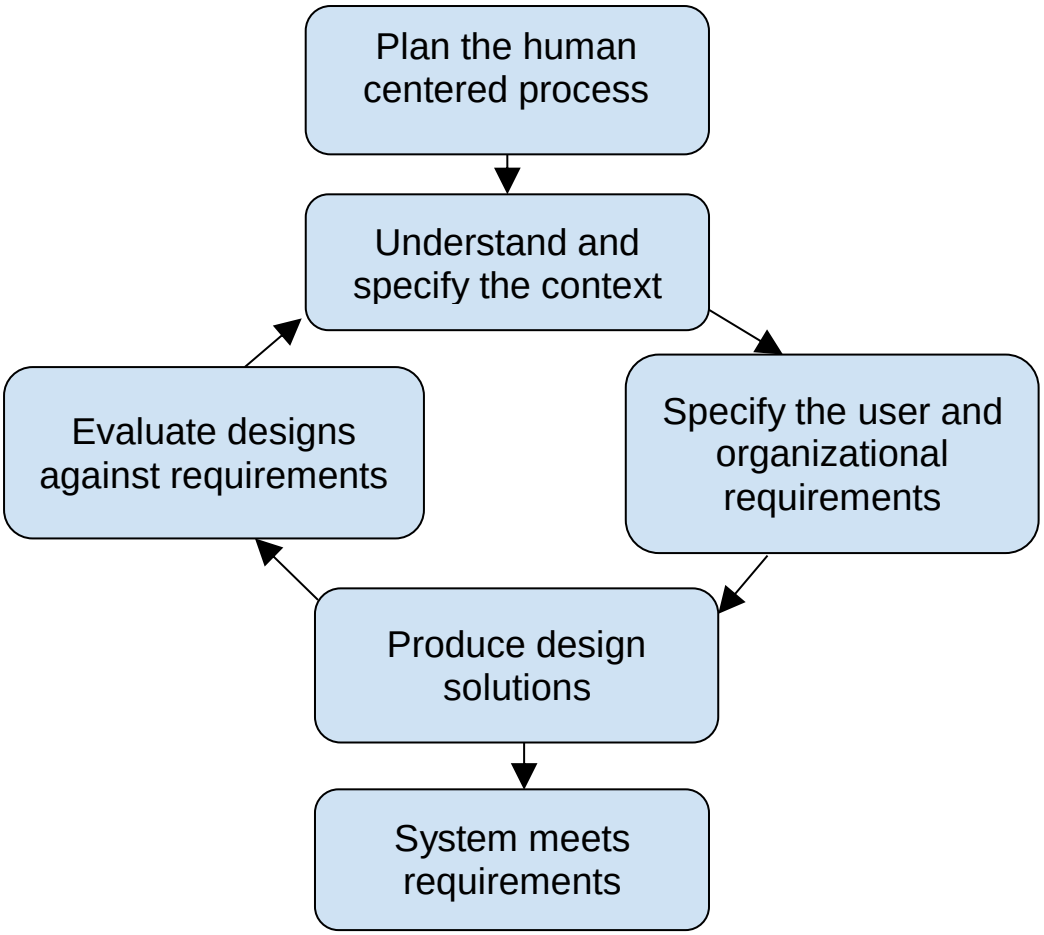


Image 2.2: general guidelines for iterative process in usability engineering in ISO standard 13407

2.2.1 Iterative Design and Prototyping

At the heart of UE is iterative design with the aim of gradually improving usability. This iterative design consists of a cycle of design, evaluation and finally redesigning based on the evaluation's results. Design and redesign are not necessarily means working on the implementation of the final product which can be hard to change and costly to redesign.

To be able to conduct evaluation, in early phases of development prototyping is used which is much cheaper, can be developed faster and be changed and modified many times. As Nielsen puts it:

"The entire idea behind prototyping is to save on the time and cost to develop something that can be tested with real users. These savings can only be achieved by somehow reducing the prototype compared with the full system: either cutting down on the number of features in the prototype or reducing the level of functionality of the features such that they seem to work but do not actually do anything." (Nielsen J. , Usability Engineering, 1993, p. 94)

A prototype can be from these forms:

- Verbal prototyping: Textual description of system interface, includes dialogues, choices and results.
- Paper prototyping: Paper sketches of the system interface which can be simple hand-drawn sketches or printouts with more details and elaborations.
- Interactive sketches: A composition of sketches which can be interactive. These sketches can be from different types, e.g. hand-drawn, HTML, PowerPoint etc.
- Working prototypes: This depends on the decision on how much functionality or features should be implemented. "Cutting down on the number of features is called *vertical prototyping* since the result is a narrow system that does include in-depth functionality, but only for a few selected features" (Nielsen J. , Usability Engineering, 1993, p. 95) and "reducing the level of functionality is called *horizontal prototyping* since the result is a surface layer that includes the entire user interface to a full-featured system but with no underlying functionality. […] Finally, one can reduce both the number of features and the level of functionality to arrive at a scenario that is only able to simulate the user interface as long as the test user follows a previously planned path. Scenarios are extremely easy and cheap to build, while at the same time not being particularly realistic." (Ibid.)

2.2.2 Iterative Design and Usability Evaluation

Each iteration of iterative design in UE involves evaluation of the designed prototypes to assess usability of interfaces. There are two major classes of methods for evaluation: *Usability test* and *usability inspection*.

2.2.2.1 Usability Test

Usability test refers to the empirical testing with real users performing real tasks with specific goals while been observed and recorded in a usability lab. The result will be analyzed to discover problems and find solutions. Andrews (Andrews, Lecture Notes: Human-Computer Interaction., 2017) summarized different usability tests as the following:

- "Thinking Aloud: Test users verbalize thoughts while performing test tasks.
- Co-Discovery: Two test users explore an interface together. Insight is gained from their conversation while performing test tasks.

- Formal Experiment: Controlled experiment, face-to-face with test users, measurements and statistical analysis.
- A/B Test: Controlled experiment on (part of) actual user population, typically (remote) web site users, with measurements and statistical analysis.
- Query Techniques: Involves interviews and questionnaires.
- Usage Studies: Usage data is collected from a small number of users working on their own tasks in their natural environment over a longer period." (Ibid. p115)

Usability tests are very informative and even irreplaceable "since it provides direct information about how people use computers and what their exact problems are with the concrete interface being tested" (Nielsen J. , Usability Engineering, 1993, p. 165) and therefore the developer can realize where exactly users do mistakes or at least have a confusion in using the system. On the other hand, there are some drawbacks in the usability test, e.g. it is very time consuming, users are stressed and have an unnatural feeling of been observed and judged specially when they use *thinking-aloud* technique, i.e. users should describe what they do and why they decide to do it out loud to be recoded. Furthermore, each user can only once participate in the test, because it is not possible to undo what a user has already learned and experienced and this increased the number of test users and that means to spend more time and money. It is hard to prepare a similar start situation for every user and the test conductor should be patient with different users and properly decide when to interfere to help a user. Although testing with small number of users is still very informative but the more users are tested, the result is more reliable. Moreover, perhaps the most important drawback of the usability test is, that there is a need for a system which is at least partly runnable and this means, it is not possible to conduct a test in the early phases of the development.

2.2.2.2 Usability Inspection

Usability inspection is a generic term for a set of methods in which usability experts or in some cases a mixture of experts and representative users - on the contrary to usability test in which real users participate - evaluate user-interfaces. Compared to usability tests, these methods cost less (e.g. much less number of experts is needed), are faster and through them deep usability issues can be analyzed and detected. Usability inspection can be conducted in early phases of development because it does not necessarily need a runnable prototype and therefore can be conducted even on a simple verbal description of a system.

2.2.2.2.1 Usability Inspection Methods

There are multiple methods of conducting usability inspections. Some of the important ones are:[3]

Heuristic Evaluations (HE)

This method, introduced by Nielsen and Molich (Nielsen & Molich, Heuristic Evaluation of User Interfaces, 1990), is conducted by a small team of usability experts who systematically analyze and evaluate interfaces based on the so called *Heuristics* i.e. a set of predefined usability principles (see chapter 3). HE is the most informal inspection method but at the same time, it is a very effective method and also cheap to conduct, very intuitive and simple to learn. HE can be done in early phases of the developments because it can be conducted on a different variety of designs from very simple prototypes like verbal description or paper mock-up to the more complex implementation of the system. For more detailed information, see chapter 3.

Cognitive Walkthrough

Developed by Lewis, Polson, Wharton and Rieman (Lewis, Polson, Wharton, & Rieman, 1992), this method is a task-oriented walkthrough of user interfaces done by an expert or a team of usability experts with a more explicitly detailed and formal procedure compared to HE. Since most users learn to work with a system through exploring, the goal of cognitive walkthrough is to imagine user's mindset with the main focus on the learnability of the system interfaces. This method helps to understand user's mindset, their goals and assumptions (compared to the developer perspective) and can be used in early phases of development because design can be any kind of prototypes, from a simple description or paper mock-up to the more complex prototypes. This method is very time consuming, requires some learning and only helps improving the learning curve of the user interfaces considering the tradeoff between ease of learning and productivity; i.e. the ease of learning can lead to less productivity in the long term for advanced users.

Formal Usability Inspection

Introduced by Kahn and Prail (Kahn & Prail, 1994) is a simplified form of cognitive walkthrough combined with HE. It can be conducted in early phases of development and uses

[3] For a comprehensive overview of these methods see Nielsen and Mack (Nielsen & Mack, Usability Inspection Methods, 1994) . For a brief summary see Loitzl (Loitzl, 2006).

strictly defined roles with specific responsibilities in a six step procedure to detect *usability defects* or characteristics of the system which hinders users to achieve their goal or make it cumbersome and unpleasant.

Pluralistic Walkthrough

In this technique (Bias, 1991) an interdisciplinary team consisting of users, developers and usability experts find and discuss usability issues in a teamwork process, and not individually like other methods. The evaluation process is based on specific scenarios and the process can be very slow because each step should be completed and discussed before dealing with the next element in the user interface.

There are other inspection methods with more specific tasks, like *feature inspection* to study usability aspect of a sequence of features used to fulfill a task, *consistency inspection* to examine consistency of different interfaces of an application of multiple applications of the same organization and *standard inspection* to study compliance of special standards.

2.3 Summary:

This chapter explained the concept of usability engineering as a systematic method involving an iterative design process to have a user friendly interface. Iterative design consists of a chain of design, evaluation and redesigning. Evaluation can be done through testing or inspection methods like HE. HE is a very important, effective and cheap inspection method in the UE especially as a part of *discount usability* (Nielsen J. , Usability Engineering, 1993, pp. 17-21). Nielsen argues that in practice recommended and more complex usability engineering processes are rarely used, mostly because it can be expensive and time consuming. So he suggests some technics to have cheap but effective discount usability. This involves a) simply to observe users doing their work without interfaces, b) using scenarios, as "the ultimate reduction of both the level of functionality and of the number of features" instead of more complex prototypes, c) using simplified thinking aloud which is done by developers with minimal training instead of special psychologists and without expensive video and audio recording. d) Finally, using HE as the most informal and cheapest inspection method which can be done with a small number of usability experts on any design and in any phase of development. (Ibid.) In the next chapter we go through the details of HE and its characteristics.

3 Heuristic Evaluation (HE)

HE is the most informal usability inspection method in which systematically, a small set of evaluators, independent from each other, analyze and judge weather elements of a user interface follow or contradict *heuristics* or established usability principles (Nielsen & Molich, Heuristic Evaluation of User Interfaces, 1990). In the following chapter we see how HE is conducted and some of the characteristics.

3.1 Performing a Heuristic Evaluation

Conducting HE involves three phases:

3.1.1 Preparation

In the preparation phase, in a prototype design of system of any type - verbal description, paper mock-up, working prototype or running system- interfaces will be present to the evaluators. Evaluators can receive general domain knowledge about the purpose of system but any information about the interface itself should not be revealed to them so they preserve an unbiased attitude toward the system. Sufficient information about the set of predefined heuristics is also a part of this preparation.

3.1.2 Evaluation

In the evaluation phase each evaluator independently - without discussion with other evaluators - analyses and evaluates interface elements in evaluation sessions. Each session of evaluation can be between 1-2 hours. It is recommended that “interface should be examined in two passes: first pass focuses on general flow, and the second on particular elements in detail.” (Andrews, Lecture Notes: Human-Computer Interaction., 2017, p. 95) Each evaluation should be described specifically and concretely (e.g. by adding screenshots), associated with a possible recommended solution and a heuristic principle which possibly is violated by the problem. All evaluation in the final list can be scored based on severity and frequency of the problems. A list of possible scores for severity rating can be as follows (Nielsen J. , Usability Engineering, 1993, p. 103):

Severity Rating	Description
0	this is not a usability problem at all
1	cosmetic problem only—need not be fixed unless extra time is available on project
2	minor usability problem—fixing this should be given low priority
3	major usability problem—important to fix, so should be given high priority
4	usability catastrophe—imperative to fix this before product can be released

A possible scale of frequency can be as the following table:

Score	Frequency
0	almost never (<1%)
1	rarely (1-10%)
2	occasionally (11-50%)
3	regularly (51-89%)
4	constantly (>90%)

Both severity and frequency can be added in one score as the score of criticality. (Andrews, Lecture Notes: Human-Computer Interaction., 2017, p. 101)

3.1.3 Merge and Report

In merge phase, a manager combines and summarizes all the founded evaluations. The problems in the resulted list can be one more time reviewed and scored, based on the severity and frequency to see what problems are more critical to be prioritized. Merging of evaluation can be a very tedious work. Different evaluators may find the same problem over and over with different descriptions and specifications. An evaluator could describe the same problem discovered from another evaluator in different parts of multiple evaluations. A merged list after review can be organized as a report. A possible form of report includes (Andrews, Skeleton Heuristic Evaluation Report, 2005):

- A summary of evaluations
- An introduction of evaluated system
- The motivation of evaluation
- Methodology
- User profiles (overview and categorizing of the users who will use the system)
- Extent of the evaluation (which part of system was evaluated?)
- Evaluators and evaluation environments, main positive finding
- Analysis of the main problems, list of found problems (resulted merge list of all evaluations)
- An appendix which includes individual evaluation logs and also set of heuristics used by evaluators.

3.2 Number of Evaluators

One of the core concepts of HE is that each evaluator will not notice most of usability issues. Several studies shows each evaluator on average only finds 35% of usability problems.[4] (Nielsen J. , Usability Engineering, 1993, p. 156) On the other hand, different evaluators find different usability problems and an aggregation of their finding can improve the results substantially. Image 3.1 shows how with the increase of the number of evaluators the proportion of founded usability problems changes and image 3.2 shows why it is optimal in the cost-benefit relation to conduct HE with three to five evaluators.

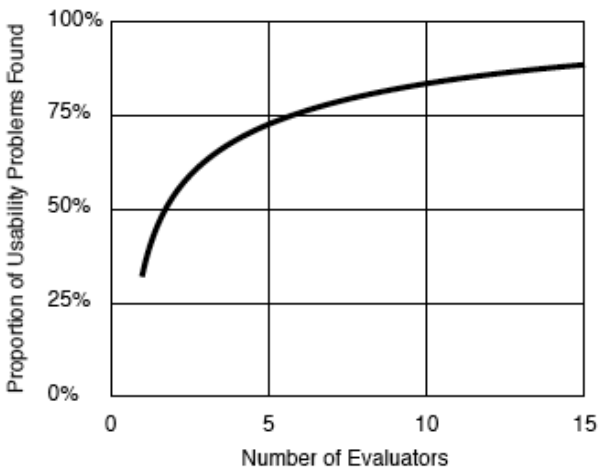


Image 3.1: Percentage of discovered usability problems
as a function of the number of evaluators (Nielsen 1993, p. 156)

[4] E.g. see Nielsen and Molich (Nielsen & Molich, Heuristic Evaluation of User Interfaces, 1990) and Nielsen (Nielsen J. , 1992).

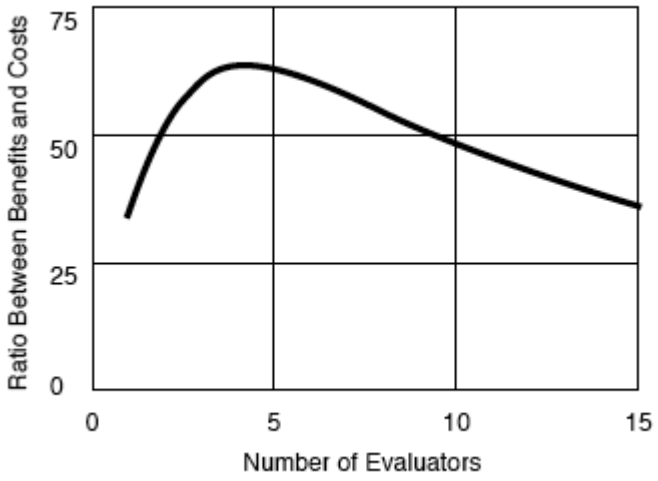


Image 3.2: The pay-off ratio for user tests as a function of the number of evaluators (Nielsen 1993, p. 174)

3.3 Effect of Evaluator Expertise

There is always a significant difference between performances of different evaluators even with the same background. This can be seen from the result of different studies in which the performance of evaluators based on the proportion of the "number of usability problems found by the top and bottom quartile (best 25% versus worst 25%) of the evaluators ranged from 1.4 to 2.2 with a mean of 1.7." (Nielsen J. , Usability Engineering, 1993, p. 163) Besides that, studies show that expedites of the evaluators influence significantly on the number of founded usability problems. (Ibid) Nielsen (Nielsen, 1992) shows not only expertise in usability evaluation is important but domain knowledge is also very beneficial. The *double experts* group of evaluators with the knowledge of both usability and domain of interface found 1.5 times more than *single experts* group and 2.7 times more than *novice evaluators*.

In conclusion, as Nielsen states, "even though heuristic evaluation can be performed by people with little or no usability expertise […], it is preferable to use usability specialists as the evaluators, and optimal performance requires double specialists." (Nielsen J. , Usability Engineering, 1993, p. 162)

3.4 Heuristics

Heuristics are general principles or as Nielsen puts it “broad rules of thumbs”, derived from experience which indicated basic characteristics of usable interfaces. (Nielsen J. , Usability Engineering, 1993, p. 115). There are different sets of heuristics which are used in UE.[5] Nielsen and Molich (Nielsen & Molich, Heuristic Evaluation of User Interfaces, 1990) defined the first nine heuristics and Nielsen (Nielsen J. , Usability Engineering, 1993) added a tenth heuristic and some principles are renamed by him (Nielsen J. , Enhancing the explanatory power of usability heuristics, 1994). The following list is according to Nielsen (Nielsen 1994): [The names in square brackets are the old names of the corresponding heuristic]

1. “Visibility of system status [Feedback]

The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

2. Match between system and the real world [Speak the users’ language]

The system should speak the users’ language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. User control and freedom [Clearly marked exits]

Users often choose system functions by mistake and will need a clearly marked “emergency exit” to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. Consistency and standards [Consistency]

Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

5. Error prevention

Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

6. Recognition rather than recall

Minimize the user’s memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another.

[5] For a summary of different heuristics see Loitzl. (Loitzl, 2006, chapter 5)

Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. Flexibility and efficiency of use [Accelerators]

Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allows users to tailor frequent actions.

8. Aesthetic and minimalist design [Minimalist design]

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

9. Help users recognize, diagnose, and recover from errors [Good error messages]

Error messages should be expressed in plain language (no codes or jargon), precisely indicate the problem, and constructively suggest a solution.

10. Help and documentation

Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large."

Very similar to Nielsen's heuristic set, there are *eight golden rules* of Shneiderman (Shneiderman, Plaisant, Cohen, Jacobs, & Elmqvist, 2016) which are also derived heuristically from experience. There are also other heuristic sets with specially defined or reinterpreted for specific domains for example rich web heuristics like *Instone heuristics* which are an interpretation of Nielsen set for the evaluation of webpages (Instone, 1997) and similarly the *Skinner and McMullin Rich Internet Application Heuristics* (Skinner & McMullin, 2003).

Muller et al. (Muller, Matheson, Page, & Gallup, 1998) extend Nielsen heuristics to be more general and usable for everyone and not only for software developers and usability experts. In *participatory Heuristic Evaluation* not only usability experts participate in evaluation but users also have a role as evaluators. Based on Floyd's[6] distinction between *product-oriented* and *process-oriented* paradigms in software engineering, they argue that Nielsen's heuristics are related to product-oriented paradigm, "which is concerned with the computer artifact (design or computer system) in relative isolation" (Ibid. p15). On the other hand "process-oriented paradigm is concerned with the computer artifact in the context of the

[6] See: Floyd, C. Outline of a paradigm change in software engineering. In Bjerknes, G., Ehn, P., and Kyng, M. (Eds.), Computers and Democracy: A Scandinavian Challenge. Gower, Brookfield, VT, 1987.

users' work processes (more broadly, life processes)", and therefore they proposed an extended list of evaluations with some new process-oriented heuristics "for a more balanced evaluation". (Ibid.)

In the following, we see their 15 heuristic set, categorized in five sets of *system status, user control and freedom, consistency and relevancy, error recognition and recovery and finally, task and work support.* (Ibid., pp. 16-18)

- System status:

1. *"System status*. The system keeps users informed about what is going on through appropriate feedback within a reasonable time.

- User control and freedom:

2. *Task sequencing*: Users can select and sequence tasks (when appropriate), rather than the system taking control of the users' actions. Wizards are available but are optional and under user control.
3. *Emergency exit:* Users can easily find "emergency exits" if they choose system functions by mistake (emergency exits allow the user to leave the unwanted state without having to go through an extended dialogue). Make their own decisions (with clear information and feedback) regarding the costs of exiting current work. Access undo and redo operations.
4. *Flexibility and efficiency of use:* Accelerators are available to experts, but are unseen by the novice. Users are able to tailor frequent actions. Alternative means of access and operation are available for users who differ from the "average" user (e.g., in physical or cognitive ability, culture, language etc.).
5. *Match between system and the real world:* The system speaks the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Messages are based on the users' real world, making information appear in a natural and logical order.
6. *Consistency and standards:* Each word, phrase or image in the design is used consistently, with a single meaning. Each interface object or computer operation is always referred to using the same consistent word, phrase or image. Follow the conventions of the delivery system or platform.
7. *Recognition rather than recall:* Objects, actions, and options are visible. The user does not have to remember information from one part of the dialogue to another.

Instructions for use of the system are visible or easily retrievable whenever appropriate.

- Consistency and relevancy:

8. *Aesthetic and minimalist design:* Dialogs do not contain information that is irrelevant or rarely needed (extra information in a dialog competes with the relevant units of information and diminishes their relative visibility).
9. *Help and documentation:* The system is intuitive and can be used for the most common tasks without documentation. Where needed, documentation is easy to search, supports a user task, lists concrete steps to be carried out, and is sized appropriately to the users' task. Large documents are supplemented with multiple means of finding their contents (tables of contents, indexes, searches etc.).

- Error recognition and recovery:

10. *Help users recognize, diagnose and recover from errors:* Error messages precisely indicate the problem and constructively suggest a solution. They are expressed in plain (users') language (no codes). Users are not blamed for the error.
11. *Error prevention:* Even better than good error messages is a careful design that prevents a problem from occurring in the first place. Users' "errors" are anticipated, and the system treats the "error" as either a valid input or an ambiguous input to be clarified.

- Task and work support:

12. *Skills:* The system supports, extends, supplements or enhances the user's skills, background knowledge, and expertise. The system does not replace them. Wizards support, extend or execute decisions made by users.
13. *Pleasurable and respectful interaction with the user:* The user's interactions with the system enhance the quality of her or his experience. The user is treated with respect. The design reflects the user's professional role, personal identity or intention. The design is aesthetically pleasing — with an appropriate balance of artistic as well as functional value.
14. *Quality work:* The system supports the user in delivering quality work to her or his clients (if appropriate). Attributes of quality work include timeliness, accuracy, aesthetic appeal and appropriate levels of completeness.
15. *Privacy:* The system helps the user to protect personal or private information […]." (Ibid.)

3.5 Summary

In HE, multiple evaluators, based on a set of heuristics, analyze and record independently usability issues in a system interface and at the end, their finding will be merged in a final report by a manager. Both, recording the evaluations and more importantly, merging them can be supported by software tools. In fact, merging such a large number of evaluations from different evaluators can be a time consuming and hard work to do, considering the intersections of the evaluations which are recorded by different evaluators independently from each other. The existing tools for HE are the topic of the next chapter.

4 Existing Tools for HE

This chapter gives an overview of the existing softwares for supporting HE. The goal of this chapter is to see the positive and negative aspects of the existing tools and how they can be an inspiration for a more powerful tool.

4.1 UX-Check

UX-Check (Gallello, UX-Check) is a Chrome Extension developed by Chris Gallello to support HE of the web-based interfaces. (Ibid.) UX-check is a lightweight tool and easy to install and use. It places a set of heuristics in the side panel next to the target website so the evaluator just needs to click on the specific element in the website to open a window like image 4.1. In this window the usability issue can be described and be assigned to a principle with a proper recommendation. The set of submitted evaluations can be overseen and edited (image 4.2) and finally be exported as HTML or DOCX.

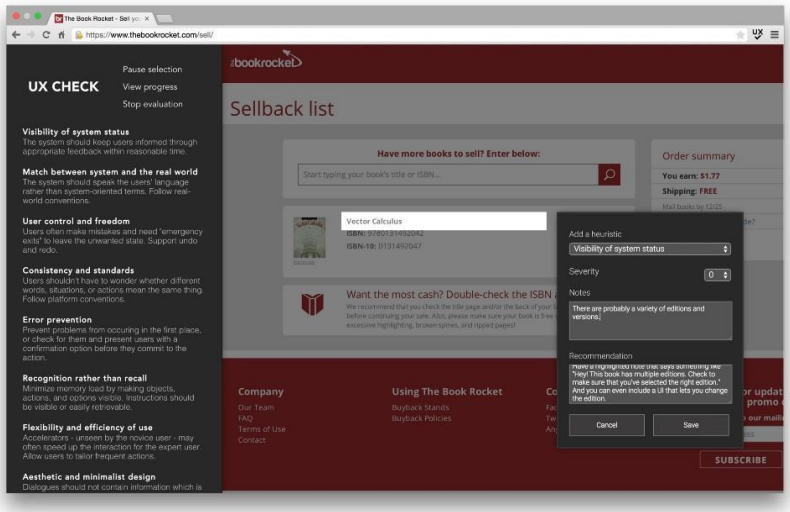


Image 4.1: Heuristic set is placed as a sidebar and after clicking
an element of the website, a dialogue opens up. (Gallello, Low cost usability testing, 2014)

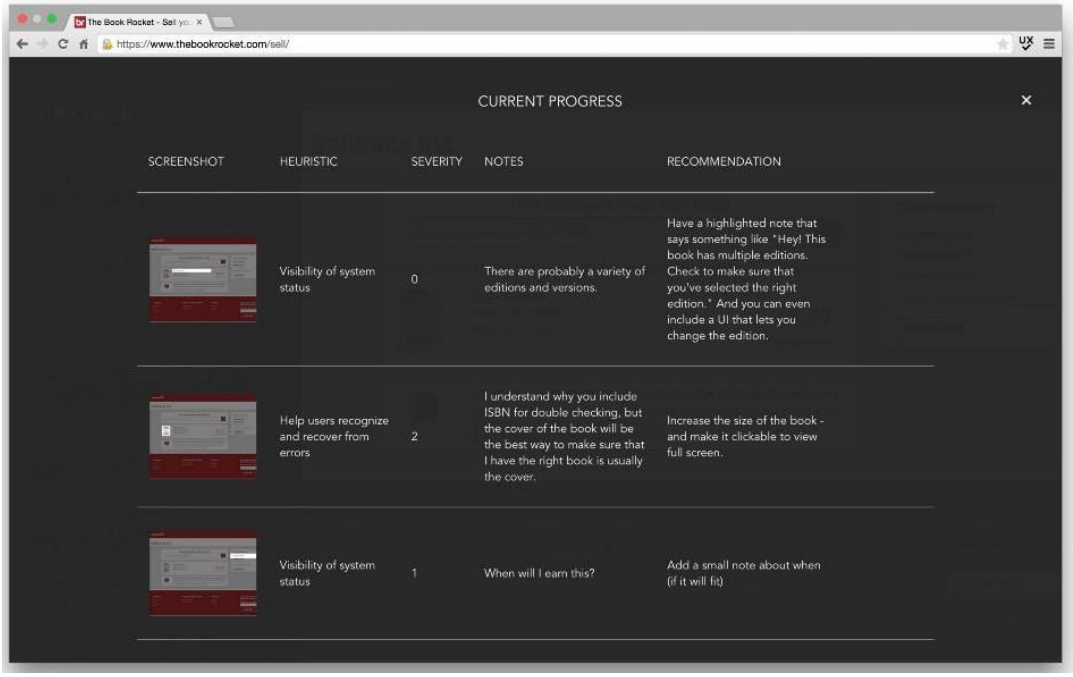


Image 4.2: Overview of the submitted evaluations in UX-Check. (Ibid.)

Being a Chrome extension makes it multi-platform, but it also means evaluation is only possible in a Chrome environment and there is no way to evaluate appearance and functionality of a website in other web browsers and considering this fact that different browsers give different user experiences for the same website, this could be a downside for UX-Check. Furthermore, being a uni-user tool there is no role as a manager defined in this tool and therefore, there is no way to compare the results and find similarities between evaluations and finally merge them.

4.2 UzReview

Similar to the Ux-Check which was a Chrome based tool for HE, UzReview (UzReview) is a Mozilla based, open-source tool developed by uzilla group, that functions as a sidebar for Mozilla for evaluation of the web-based interfaces. After defining a project, unlike Ux-Check, which evaluations were based on screenshots, in UzReview logging of heuristics is based on a URL or a keyword (see image 4.3). Beside all the limitations of Ux-Check, like being restricted to one specific browser representation of a website (in Mozilla) and being a uni-user tool, this tool is not supported anymore and cannot be installed on the new versions of Mozilla.

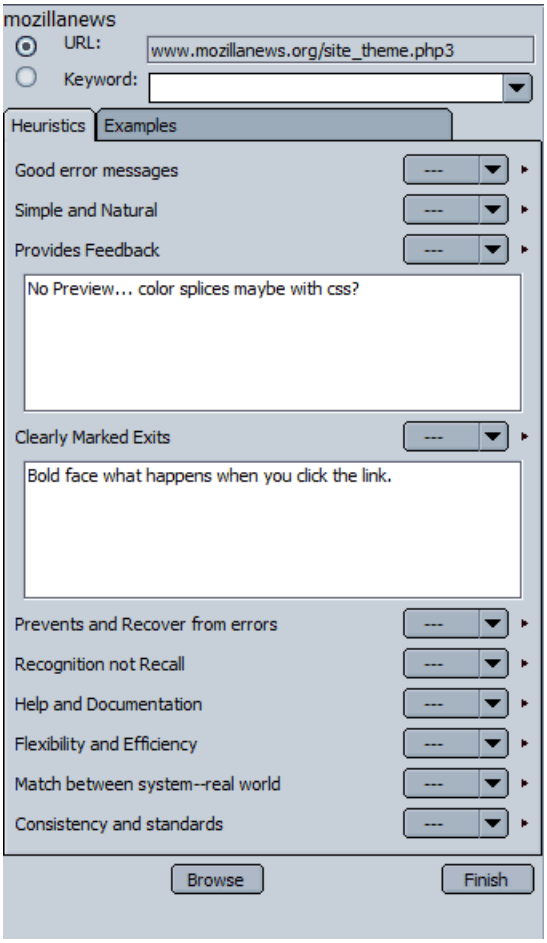


Image 4.3: Logging evaluations based on URL or keywords. (Ibid.)

4.3 Heuristic Evaluation Manager (HEM):

Heuristic Evaluation Manager or HEM (Loitzl, 2006) is an online collaborative web-application for HE, developed by Martin Loitzl as a master dissertation for the University of Graz. HEM is implemented in PHP4 and XHTML as a static web-application. Evaluators can have an account and submit their evaluations supported with screenshots (see image 4.4 and 4.5). The manager can finally produce a merge list of all submitted evaluations and produce a report (image 4.6). HEM is a multi-user application which allows collaborative evaluation necessary in HE. It is also not restricted in a specific type of HE, e.g. being only for websites like UX-Check.

On the downside, merging phase in HEM could be very difficult, considering this fact that the app does not provide any help to recognize similarities in the submitted evaluations, and therefore the manager should read all the evaluations and compare them to make an optimal merge report which can be a tedious work to do. In addition to that, each evaluation

is only based on screenshots without any other keywords, e.g. place of the evaluation to help make a quick skim of the differences and similarities between them. HEM is not anymore available to use and despite all of the mentioned downsides, it was a very good example of how a HE tool should be. HE is a collaborative work and in contrary to the other tools, HEM targeted this feature and with some improvements in merging phase and possibility of adding some details in each evaluations (e.g. place, link) could be a good model of how a HE tool should be designed. In my work, I use the HEM as a base model and try to improve its negative aspects, mainly bringing some improvements in the merge component.

Image 4.4: HEM is a multi-user application. (Ibid, p 146)

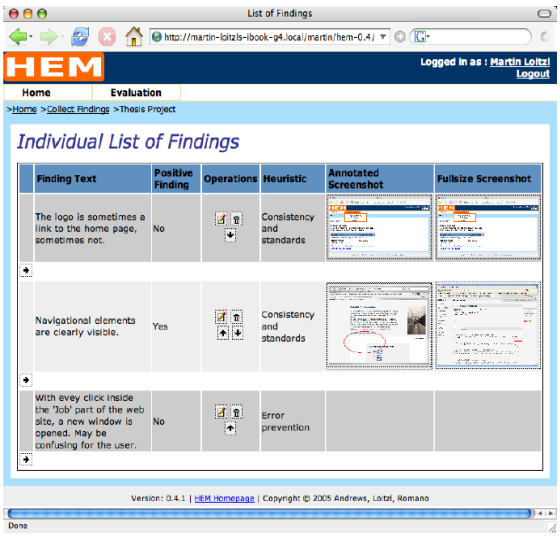


Image 4.5: Evaluation fields in HEM. (Ibid, p 143)

Image 4.6: Merging desktop in HEM. (Ibid, p. 144)

4.4 Summary

After analysis of the existing tools for HE, one can conclude that any useful tool for HE, should be a) a multi-user application which makes it possible for a collaborative evaluation of a system (any system and not only websites) and also b) be smart enough to recognize similar evaluations to recommend possible merges in the final report. All the existing examples of tools for HE lacks one or both of the mentioned characteristics. In the next chapter we will see a prototype design of SHE or *smart heuristic evaluation* as a possible tool for supporting HE and fulfilling both mentioned characteristics.

5 Concept and Prototype Design of Smart Heuristic Evaluation (SHE)

Inspired from the existing tools for HE, the goal of this section is to concept and prototype *SHE (Smart Heuristic Evaluation),* a collaborative web application tool which makes it possible for *multiple users* to participate in a *project* and *evaluate* a system, similar to HEM (Loitzl, 2006), and additionally, be smart enough to *recognize similar evaluations* and recommend them to the manager to *merge* them in the final *report*. Based on these goals, SHE is designed to separate different tasks in different components working fairly independent from each other.

5.1 Components of SHE

There are three main components in SHE: *User manager*, *project and evaluation manager*, *merge and report manager.* These three components are responsible for different goals which SHE should fulfill. Each component follows MVC[7] or Model-View-Controller architecture. In the following, mostly two layers are presented in each component: the view and the model. Except the basic logic and algorithms in the merge and report component, the controller layer mostly depends on the framework used in SHE (Django) and will be discussed in the chapter 6 and 7.

5.1.1 User manager

This component is responsible for signing up/in and authentication of users. The goal is to separate all general tasks related to the evaluation and merging process from user management tasks e.g. signing up, signing in, password recovery, profile management etc. This component has these views:

1. Signing up: In this view a user can for the first time register and create an account in SHE. A name, an email and a password is necessary for this stage (image 5.1).
2. Signing in: Registered users can sign in, using their email and password (image 5.1).

[7] MVC or *Model-View-Controller* introduced first as a framework by Trygve Reenskaug for the Smalltalk as a platform in the late 1970s and since then, evolved to be one the main design patterns in software development. For more on MVC see: Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal [1996]. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley and Sons. Inc., New York, NY, USA. ISBN 0471958697.

3. Password recovery: In case of forgetting password it is possible to reset it using the registered email (image 5.1).
4. Profile view: User can have a profile with a picture and a short bio (Image 5.2). This profile can be edited (image 5.3).

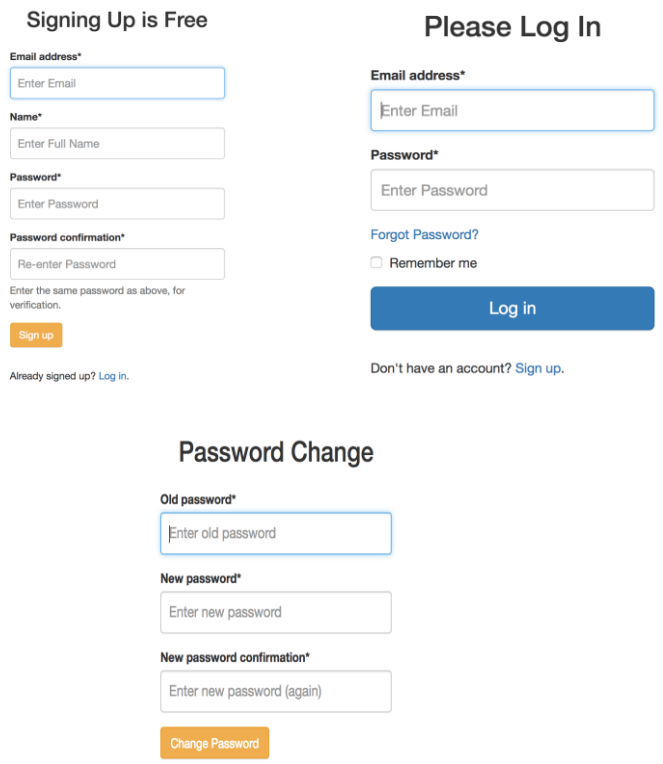


Image 5.1: Views for signing up/in and changing password in SHE

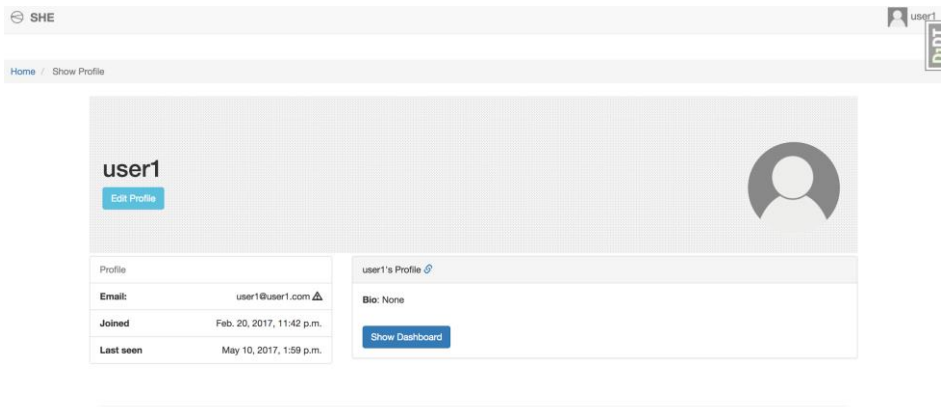


Image 5.2: View of user profile

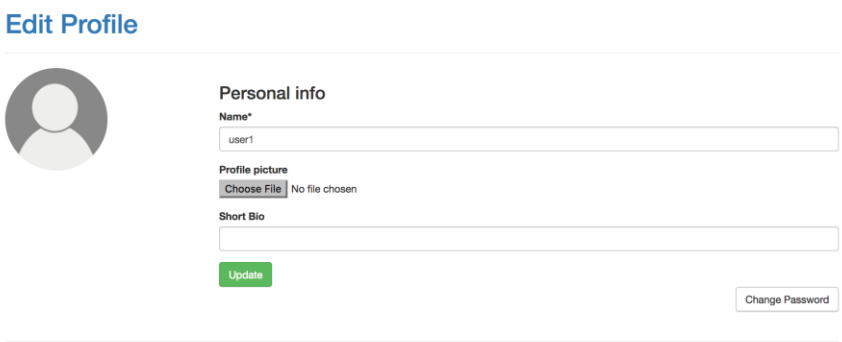


Image 5.3: View for profile edit

5.1.1.1 Model in the User Manager Component

A user signs up with a name, an email and a password. A user also can have an optional profile with a photo and a short bio. There are some other data related to the status of the user which can be optional and used to manage user access as an admin. Image 5.4 shows an overview of the model in user manager component. The model in this component is based on features of edge v2.2. (Edge2.2)

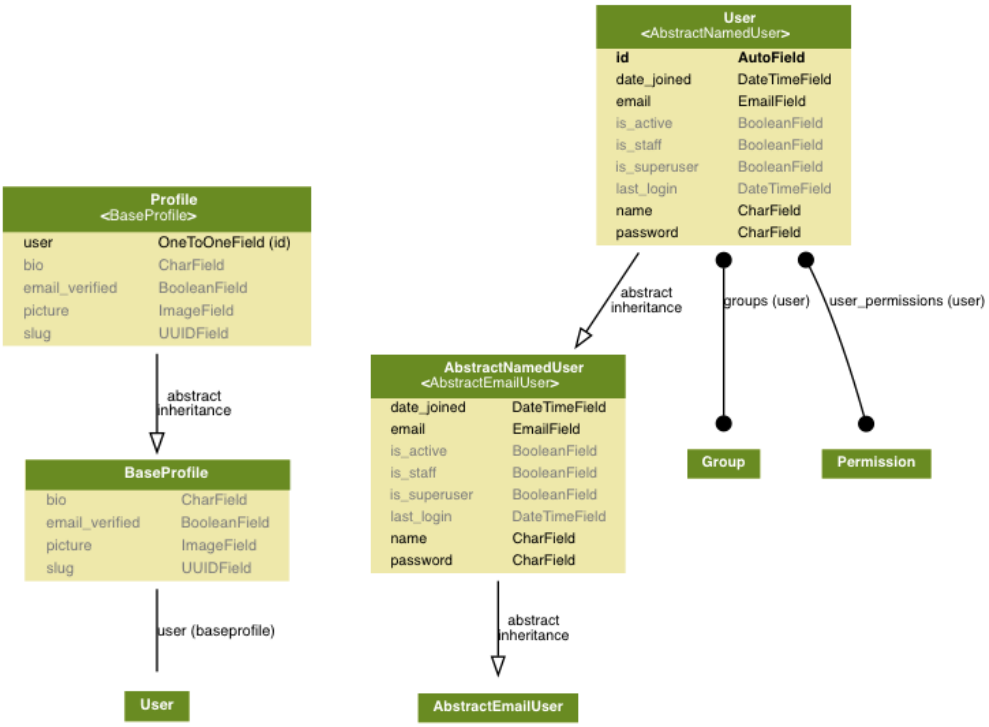


Image 5.4 : Class diagram of the model in user manager component

5.1.2 Project and Evaluation Manager

This component makes it possible for a user to be a manager and define a project or as an evaluator submit evaluations for a project. The views in this component are:

1. Dashboard: In the dashboard each user has an overview of all the projects in which he/she is a manager or evaluator (image 5.5), a table for all of the heuristic sets which are defined by the user and also a table of all the defined environments in which the user wants to evaluate a system. In the dashboard user can have some information about each project, like the name of the project, its description, status of the project (deadline of the evaluation is passed or not), number of evaluators, number of submitted evaluations and some operations for each project based on being a manager or only an evaluator.
2. Defining a new project: A user can be a manager and define new projects (image 5.6). A project has a name, possibly a link, a set of heuristics, description of the project, a deadline for evaluation and also some evaluators (manager is per default an evaluator).
3. Defining a new heuristic set: Each project should have a heuristic set according to which evaluations are submitted. Every user can use the already defined heuristics in their projects or define new ones. Image 5.7 shows a possible interface for this task. Each set has a title, a description and multiple principles. Each principle has a title and a description.
4. Project detail: After clicking on the name of each project in dashboard, the user can see the details of that project based on being manager or evaluator. Image 5.8 shows the project detail for evaluator and manager respectively. Beside project information like title, description and deadline, an evaluator sees only his/her submitted evaluations. But a manager sees also all the submitted evaluations and merged evaluations (if any available).
5. Defining evaluation environment: Evaluators can define specifics of their evaluation environment. Each environment contains age, gender, web browser type, operating system, monitor size, monitor resolution and other information related to the environment. Each evaluator can define multiple environment sets. (Image 5.9)
6. Evaluation form: An evaluator can use an evaluation form before the deadline of a project and submit any number of evaluations. Image 5.10 shows an evaluation form. Each form has these fields: title, place (where in the system this evaluation refers to),

link (if any), tags (keywords of the evaluations), heuristics (principles which are violated or followed), description, recommendation (possible solution), positivity (whether the evaluation is positive or negative), severity, frequency and finally a screenshot.

7. Edit views: Each project, evaluation and heuristic sets also can be edited. Edit views in each case are the same as the forms which are used for the defining of each project, evaluation or heuristics.

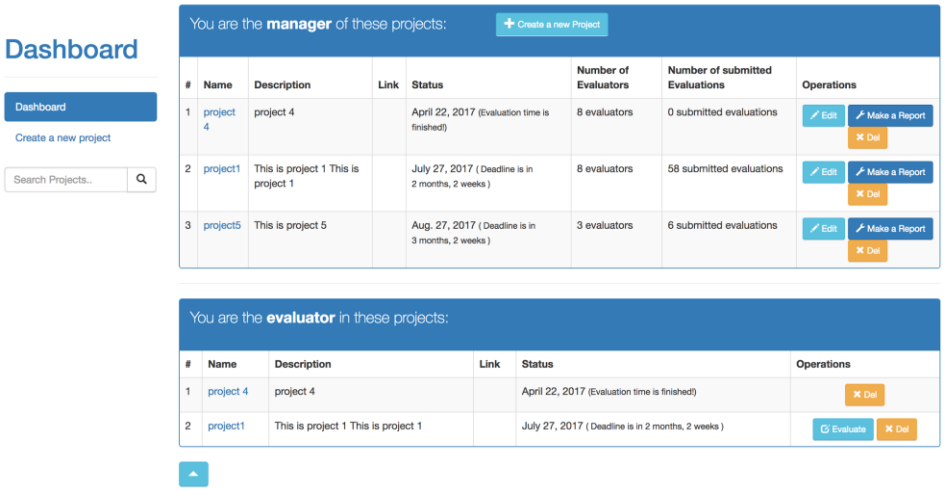


Image 5.5: Dashboard view gives an overview of all the projects of a user

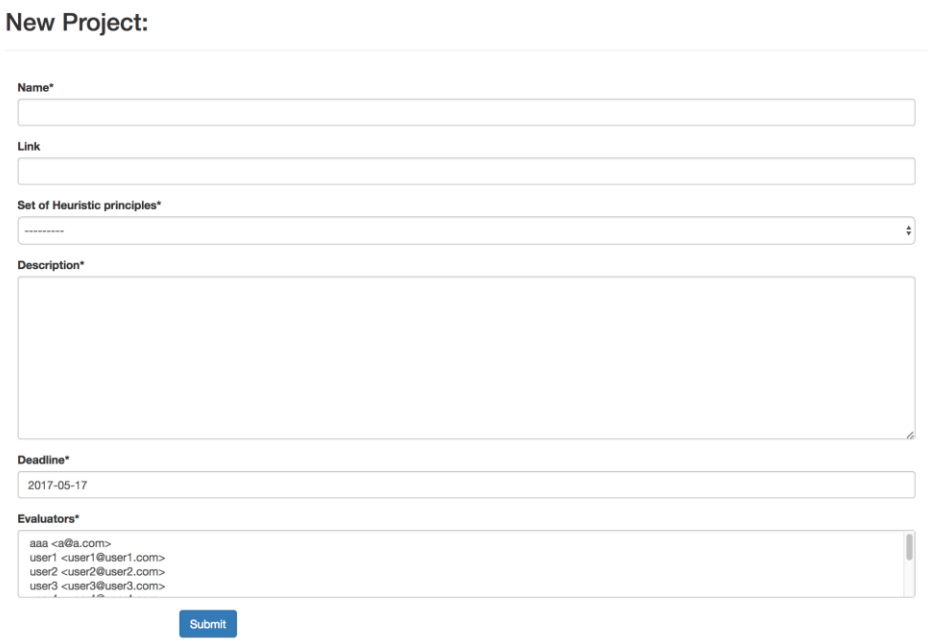


Image 5.6: Form for defining a new project

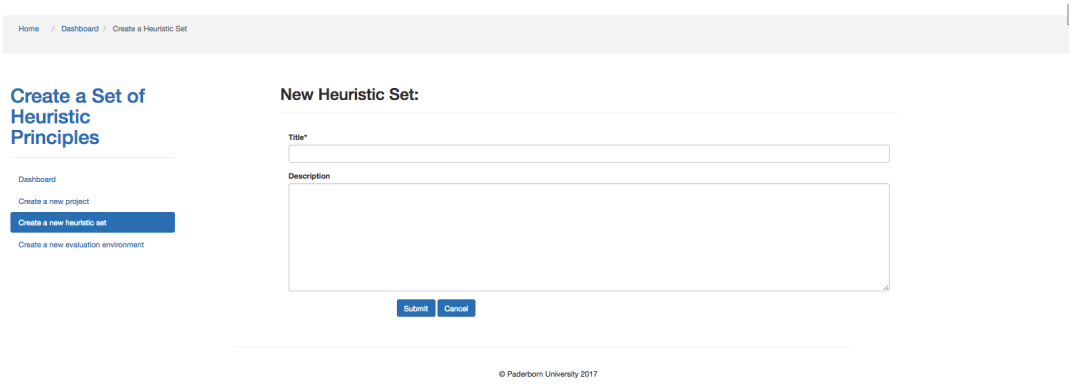


Image 5.7: Defining a new heuristic set

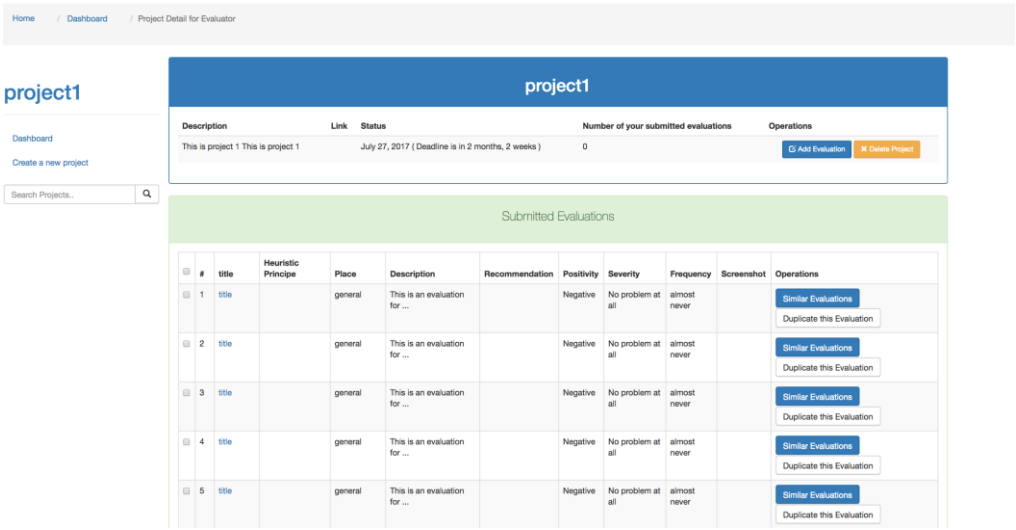


Image 5.8: Project detail view

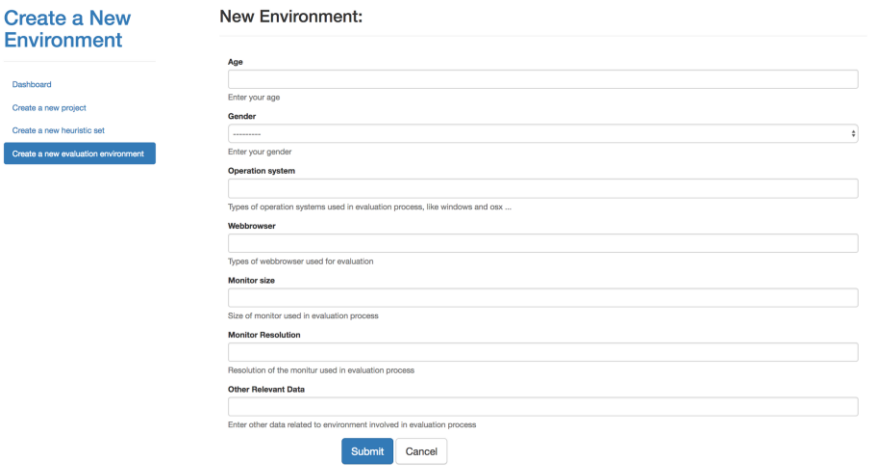


Image 5.9: Creating a new environment

Image 5.10: Form for creating a new evaluation

5.2 Model in Project and Evaluation Manager

Image 5.11 shows a class diagram based on the database structure of the *project and evaluation manager* which has six classes: *Project*, *Evaluation*, *SetOfHeuristics*, *HeuristicPrinciple*, *Screenshots* and *Environment*.

Each *Project* has a name, description, a manager (from class *user*), a set of heuristics (from class *SetOfHeuristics*), a creation time, a deadline and finally a link (optional). A project can also have multiple evaluators (from class *user*) and multiple evaluations (from class *Evaluation*). An *Evaluation* belongs to a project (*ofProject* field) and has an *evaluator*. Other fields of this class are:

- *Title*: Title of the evaluation.
- *Place*: Where this evaluation refers to.
- *Link*: A possible link (e.g. if evaluation refers to a page in a website).
- *Positivity*: Shows if an evaluation is positive or negative.
- *Description*: The explanation of the issue.
- *Recommendation*: Evaluator uses this field to suggest some solutions to the issue.
- *Frequency*: How often this issue repeats itself (see section 3.1.2).

- *Severity*: The level of criticality or usability importance (see section 3.1.2).
- *Tags*: Keywords of this evaluation.
- *Merged*: An evaluation can be merged or not merged (normal evaluation). The merged field is a *boolean* field which is *true* if the evaluation is merged and *false* if the evaluation is normal or not merged.
- *Screenshot*: If an evaluation is a normal evaluation has a screenshot or in case of being merged has multiple screenshots (from class *Screenshots*).
- An *Environment* has information of the creator (a user) and some fields to specify monitor resolution, monitor size, operation system and other information regarding the evaluation environment. Each user can define multiple environments.
- A *SetOfHeuristics* has a creator (a user), title, description and multiple principles which are from the class *HeuristicPrinciples* .

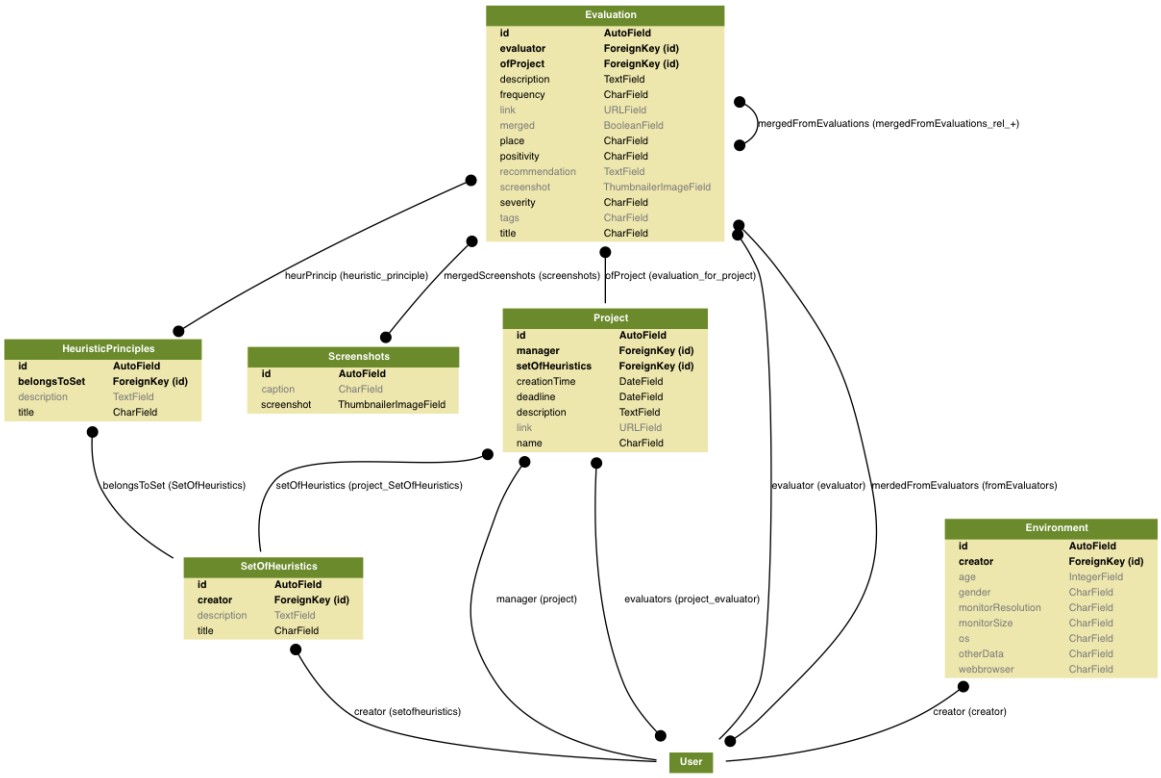


Image 5.11: Class diagram of the model in the project and evaluation manager

5.2.1 Merge and Report Manager

This component makes it possible for a user, who is the manager of a project, to have an overview of all the submitted evaluations, to merge similar evaluations and produce merged evaluation and finally make and export reports. This component has these views:

1. Defining merged evaluation: Each merged evaluation similar to normal evaluation has title, place, link, heuristic principles, description, recommendation, positivity and screenshots. Additionally, a merged evaluation has the name of evaluators who found this evaluation, an average score for the frequency and severity based on the submitted scores of different evaluators.
2. Merge desktop: In this view a manager can merge normal evaluations and create merged evaluations. The manager can have an overview of all evaluations in a column categorized by their evaluators, and after choosing each evaluation, a recommendation engine finds similar evaluations and presents them in another column. The manager can choose which evaluations should be merged. The result of merging will be presented in a merged evaluation which can be edited and summarized (see image 5.12 and 5.13)
3. Making report: After the merging phase, a report can be exported. A report can include these parts: Title and description of the project, evaluators list and their environments, list of positive and negative merged evaluations. Finally, as an appendix, a list of all the submitted evaluations and used heuristics and their description.

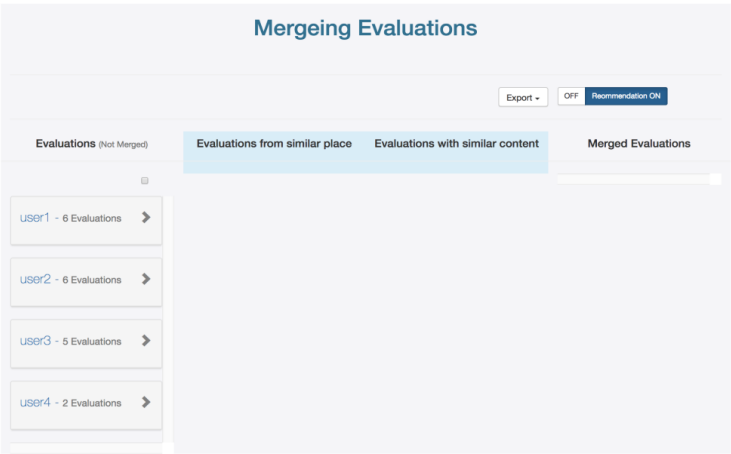


Image 5.12: Merge desktop for merging evaluations

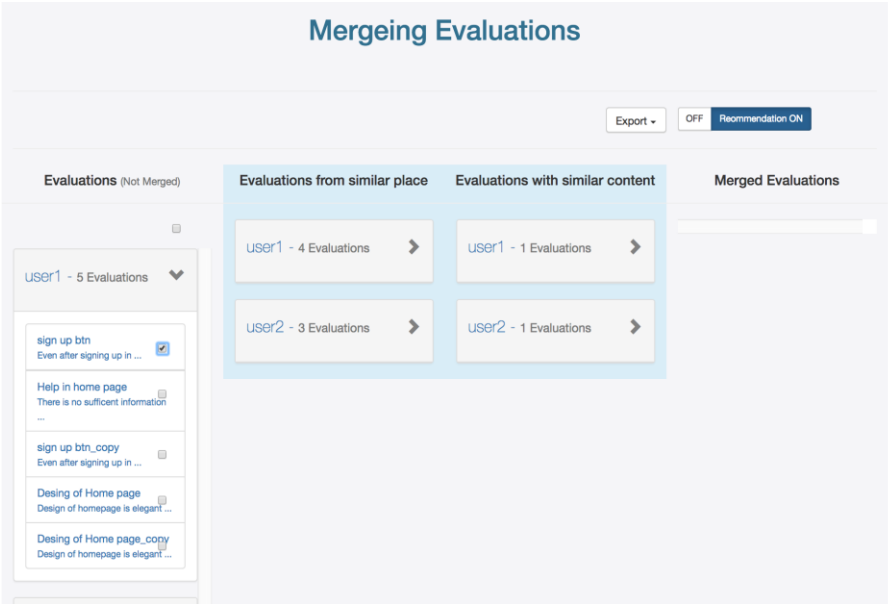


Image 5.13: Merge desktop shows similar evaluations for a chosen evaluation

5.3 Building a Recommendation Engine Using Data Mining Methods

One of the main goals of the SHE is to makes it easier for the manager to merge submitted evaluations and the merge and report component is responsible for this task. The recommendation engine in SHE uses two main aspects of evaluations, *place* and *content*, to find similarities between them. Based on these two aspects, there are two sorts of recommendations in SHE: *Place-based* and *content-based recommendation*. Data mining as "the process of collecting, searching through and analyzing a large amount of data in a database to discover patterns or relationships" (data mining definition in dictionary.com) provides methods to analyze evaluations and find similarities between them.

5.2.2 Place-based Recommendation

In the design of each evaluation object, there are two fields which have place-relevant information: One field is *place* and the other is *link*. The field *link* is only relevant in the evaluation of a website project and indicates the page which an evaluation refers to. This link is unique and cannot be expressed in different ways. Therefore, to find all evaluations with the same given link, a simple database query will do this job. But a place-based query needs to be done in a more sophisticated way than a simple search in a relational database. Each user may use different combination of words to describe where the evaluation refers to.

Therefore, information retrieval methods for text analysis and building search engines should be deployed.

A common method for doing this task would be to count the words in each document (here each evaluation) and build a key-value dataset by mapping words as a value to the number of occurrences as a key. For example the key-value dataset for “Cat cat dog” would be:

```
{“cat” : 2 ; “dog” :1}
```

In the next step, the number of occurrences should be converted to a score of importance or “weight” to find which document is more relevant to the given query. One possible example of scoring models[8] is *TF-IDF* or *term frequency - inverse document frequency*. TF-IDF is higher proportionally if a word in query frequently appears in a document unless this word is a stop word which is one of the most common words in a language. TF are defined as follows:

$$tf_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Where tf_{ij} is the frequency of term i in document j normalized by the maximum number of occurrence of any term in the same document. (Rajaraman & Ullman, 2011, p. 8)
On the other hand where term i appears n times in N document IDF is defined:

$$idf = log_2(N/n_i)$$

And the TFIDF score is the multiplication of both terms:

$$tfidf = tf_{ij} \times idf_i$$

“The terms with the highest *TF-IDF* score are often the terms that best characterize the topic of the document”. (Ibid.)

5.2.3 Content-based Recommendation

Content-based recommendation is done based on text analysis of the following fields in each evaluation: *Description, recommendation, tags*. Each of these three fields in each evaluation are analyzed to compute features like *TF-IDF*. Then for a given evaluation the algorithm searches through all the evaluations and computes distance between the specified features like *TF-IDF* and find *k*-evaluations with the most similarity - or less distance - and return them as the recommendation list for a possible merge. To find similarities, there is a

[8] There are multiple scoring models like *Pivoted length normalization, BM25 , Query likelihood, PL2*. For more on this see: Hui Fang, Tao Tao, and Chengxiang Zhai. 2011. Diagnostic Evaluation of Information Retrieval Models. ACM Trans. Inf. Syst. 29, 2, Article 7 (April 2011)

number of distance functions that can be used (e.g *Consin*, *Jaccard*, ...)[9]. One can specify *k* as a fix number to get a number of evaluations or give a threshold for the distance function and as a result *k* changes based on the number of evaluations which is in the range of the distance limitation. The algorithm used to find similar evaluations to a given query evaluation is the simplest form of the *nearest neighbor search (NNS)* algorithm[10] and can be described as follows:

```
    Input: Query evaluation e_i
    Output: List of k evaluations
    Simple NNS Algorithm:
-   Search over each all evaluations e_j in a project
                •  Compute dis = distance(e_i , e_j )
-   Return k evaluations with the smallest dis.
```

The running time of this algorithm is *O(dN)* with *N* is the number of evaluations and *d* is the dimensionality of query evaluation.

5.3 Summary

SHE has three main components: 1) user manager, 2) project and evaluation manager and 3) merge and report manager. The architecture of SHE is in MVC which means each component has three separate layers of model, view and controller. Merge and report manager uses data mining methods for analysing evaluations and finding similarities between them based on two categories of data fields to make two kinds of recommendation: *place-based* and *content-based* recommendation. The next chapter presents some technologies which are used in the implementation of SHE.

[9] This article introduced 12 similarity functions (Abbasifard, Ghahremani, & Naderi, 2014).

[10] For an overview of some of the other NNS algorithms and pro and cons of the simple algorithm see Abbasifard and others (Abbasifard, Ghahremani, & Naderi, 2014).

6 Framework and Technologies used in SHE

This chapter goes through some technologies which are used in the implementation of SHE.

6.1 Python

The whole backend of SHE is written in Python. Python is a widely used programming language which uses a simple and minimalistic syntax, as well as a simple and coherent structure and despite that, still manages to be a multipurpose and powerful language. In 1991, Guido van Rossum, a Dutch mathematician and computer scientist, published the first version of Python, a language based on the design philosophy of “simplicity and readability”. (Van Rossum, 2009) Since then, Python continues to grow as a free and open source project and now is the fourth most used language (The 2015 top ten programming languages, 2015) and continues in its way up based on the fact that Python grew the most in popularity in the last five years (Pypl popularity of programming language, 2016).

Python is a *high level*, *general purpose*, *multi-paradigm*, *interpreted* and *dynamically typed* language. As a *high level* language, Python has a strong abstraction from low level and hardware related details (e.g. memory management) and therefore is closer to the human language than to the machine language. Being a *general purpose* language, Python's application domains are not limited to a specifics task. Although Python is famous as a powerful scripting language, a reputation based on its ease of use, interactive mode of execution and high level of embeddability with other languages like C and Java (Lutz, 2013, p. 13), Python has a successful record of use in different domains like dynamic web applications, desktop GUIs, system programming, prototyping, database programming, numeric and scientific programming, gaming, etc. Python is a *multi-paradigm* language and supports procedural, declarative, object oriented and functional programming paradigms. Being an *interpreted* language means the execution of a Python program involves an interpretation process in which - completely hidden from the programmer's perspective the source code of a Python program is read and translated by a program called “interpreter” which results in producing *byte code* which is different than machine code produced in compiled languages. *Byte code* is a low-level, Python specific but platform independent (and therefore portable) representation of the source code. The translation process is only repeated when the source code or Python version changes. (Lutz, 2013, pp. 30-32) Python is

dynamically typed and therefore type checking happens at runtime and not at translating time. That means, there is no need to declare a type for a variable before using it. This feature makes Python a very flexible and concise language, but at the same time has some drawbacks like lacking early feedback, which is normal in static typed languages. (Python docs)

Beside high readability of python code, one of the most important reasons to use Python is the high developing productivity resulted from the minimalist syntax and a very strong standard library following a philosophy called "Batteries included" which means the standard library should be strong enough that the user does not need to use third party libraries unless he or she chooses to use them (like buying an external battery for a digital device). (Lutz, 2013, pp. 3-4)

6.2 Django

Django (Django Project) is high-level Python web framework originally developed at *World Online*[11] now run by an international team of volunteers as an open source project. Django designed to be fast, secure and scalable and is the most popular Python web framework (Python - Web framework rankings). Developing based on Django can be very fast, not only because of simplicity and minimalistic aspects of Python, but also because Django takes care of common features like user authentication, content administration, form handlig etc. With its user authentication system which provides a secure way to manage user accounts and passwords and also form handling tools, Django helps developers avoid many common security mistakes, such as SQL injection, cross-site scripting, cross-site request forgery. (Django docs)

6.2.1 Django Architecture

Image 6.1 shows an overview of the Django architecture. Django represents the green square which works in the middle of the web browser and a database. The URL dispatcher uses a "Python module informally called a *URLconf* (URL configuration). This module is pure Python code and is a simple mapping between URL patterns (simple regular expressions) to Python functions (views)". (Django docs)

[11] The web department of *ljworld* in Lawrence, Kansas, USA.

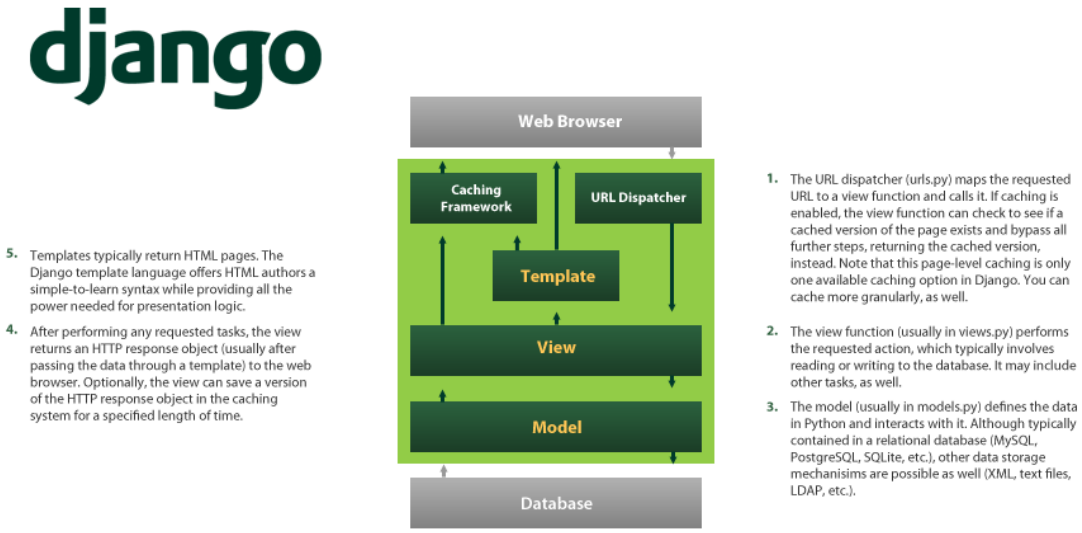


Image 6.1: Django architecture (Ibid.)

For example the following code is a simple *URLconf* which takes a request to the url "*hello/*" and refers it to a *hello* function in the view module:

```
// module url.py

from django.conf.urls import include, url

from mysite.views import hello

 urlpatterns = [
    url(r'^hello/$', hello),
]
```

View layer encapsulate the logic which is responsible for processing the user's request. This process can lead to reading and writing data which means contacting a database through model layer which is an abstraction layer for the structuring and manipulating data. (Django docs) The simple *hello* function in *view.py* module is like the following:

```
// module view.py

def hello(request):
        return HttpResponse("Hello world")
```

The Model is the interface of database and defines the fields and the behavior of data. The following code shows how a simple model in Django is implemented (Django docs):

```
// module model.py

from django.db import models
class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

This class of python is equal to write the following SQL code:

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```
[12]

The above example shows that there is no need to write SQL code in a Django application and database can be managed based on pure Python modules.

6.3 Graphlab Create

Graphlab Create is an open source Python library, written in C++, for quickly building large scale and high performance data products. (Graphlab Create) The Graphlab project was started by Prof. Carlos Guestrin of Carnegie Mellon University in 2009, originally designed for Machine Learning tasks but developed to a be powerful in a full range tasks in data-mining, data-visualisation, deep learning, network analysis etc., out-performing similar libraries. (Low, Gonzalez, Kyrola, Bickson, Guestrin, & Hellerstein, 2012) SHE uses Graphlab for text analyzing and building a recommendation engine which recognizes a similarity between submitted evaluations from different users.

6.4 Django Edge

Edge is a project skeleton on top of Django which can be used as a base for multi-user web applications. (Edge2.2) Edge takes care of most of the repeated tasks in a multi-user application and prevents “doing the same changes again and again while starting a new project”. (Ibid.) The user management component in SHE is based on the Edge skeleton.

6.5 JQuery and Bootstrap

Front-end of SHE is based on Bootstrap (Bootstrap) and also uses extensively jQuery (jQuery). Bootstrap is a very - if not the most - popular HTML, CSS and JS framework for developing responsive, mobile first projects on the web, developed by Mark Otto and Jacob Thornton at Twitter. JQuery is a fast, small and feature-rich cross-platform JavaScript library

[12] Ibid.

which is free and open-source and works across a multitude of browsers. (Ibid.) JQuery is by a large margin the most popular JavaScript library used in web. (Libscore)

6.6 Summary

Choosing Django as the main framework for SHE has the benefit of using Python as the only language for the backend. This means all the interaction with the database and also all the logic layer is written in pure Python without any need of integration with other technologies. Python is designed to be simple, minimalistic, powerful and at the same time readable with a very rich standard library and also has the possibility of using third party libraries like Graphlab which is used for data mining tasks in SHE. Next chapter presents some implementation aspects of SHE.

7 Selected Implementation Aspects

This chapter goes through some implementation aspects of SHE. These aspects mostly cover selected main issues like the implementation of MVC architecture in SHE, the merging process and finally the implementation of recommendation engine in merge and report component. This coverage does not intent to be comprehensive.

7.1 Architecture of SHE

SHE is a Django based web application and therefore is based on a MVC architecture. The three layers of Django architecture, Model, Template and View layers (MTV), correspond to Model-View-Controller (MVC) pattern (Model → Model, Template → View, View → Controller). This twist in the name is based on an interpretation of MVC, in which "The *view* describes the data that gets presented to the user. It's not necessarily how the data looks, but which data is presented. The *view* describes which data you see, not how you see it." (Django docs) Each component in SHE is implemented in separate apps and each app has its own MVC layers. To see how each layer is implemented some code examples are presented in the following.[13]

7.1.1 Model

Model is the interface of database and defines the fields and behavior of data. In Django and (therefore in SHE) each table of data is represented in a Python class. These classes are in a file with the name *model.py* which is a Python module. For example *project* class in the *project and evaluation manager* component in SHE is implemented as follows:

```
1. class Project(models.Model):
2.     manager = models.ForeignKey(User)
3.     setOfHeuristics = models.ForeignKey
               (SetOfHeuristics,related_name='project_SetOfHeuristics' ,
                       verbose_name='Set of Heuristic principles')
4.     evaluators=models.ManyToManyField(User,related_name="p_evaluator")
5.     name = models.CharField(max_length=40)
6.     link = models.URLField(blank=True)
```

[13] In the source code of SHE, the *user management* component is implemented in three separate apps: *auths*, *account* and *profile* apps. *The project and evaluation* component is implemented in one app with the name *HE3* and finally the *merge and report* component is also in a separate app with the name *Merge*. For more on the role of apps in Django see (Django docs)

```
7.     description = models.TextField()
8.     creationTime = models.DateField(default= utils.timezone.now)
9.     deadline=models.DateField(default= date.today()+timedelta(days=15))
```

Class *project* defines a table in the database the same way the command *create table* in SQL creates tables in a relational database. Each defined field in *project* is the name of a column in the project table. To get a specific project the following query is used:

```
test_project = Project.objects.get(name= 'test_project')
```

Or for getting all the environments used by evaluators in a project ordered by the name of their creator such a query is used:

```
environments = Environment.objects.filter(creator__in =
project.evaluators.all()).order_by('creator')
```

7.1.2 View

View layer in Django is represented in *templates*. “A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted.” (Django docs) Different template engines can be used in Django. SHE uses *Jinja2* (Jinja2), one of the most popular template engines for Python. For example to create a form view for the project class such a template is used in SHE:

```
{% extends 'HE3/baseHE.html' %}
{% load staticfiles %}
{% block titleOfPage %} Create a Project {% endblock %}
{% block createProjectActive %} active {% endblock %}
{% block breadcrumb %}
   <li> <a href="{% url 'profiles:dashboard:user-dashboard'     %}">Dashboard
       </a>
   </li>
   <li>
       <a class="text-muted" href="{% url 'profiles:dashboard:project_create'
       %}">
       Create a Project
        </a>
   </li>
{% endblock %}
{% block content %}
   <div class="col-sm-9">
        <div class="container" >
               <div class="page-header">
                     <h2>New Project:</h2>
              </div>
                <div class="panel-body">
               <form class="from-horizontal" action=""
                     method ='post' enctype="multipart/form-data">
                   {% csrf_token %}
```

```
                    {% load crispy_forms_tags %}
                            {{  form | crispy }}
                  <div class="from-group">
                      <div class="col-sm-offset-2 col-sm-10" >
                          <button type="submit"
                         class="btn btn-primary"> Submit
                  </button>
                          <a class="btn btn-default"
      href= "{% url 'profiles:dashboard:user-dashboard'%}">
                                  Cancel
                         </a>
                      </div>
                  </div>
              </form>
          </div>
      </div>
  </div>
{% endblock content %}
```

This HTML code extends a base HTML code with the name *baseHE.html* and changes specific blocks e.g. **`titleOfPage`** of **`breadcrumb`** and finally include a form template to present variable *form* received from controller layer (from the module *view.py*). The *form* variable has all the HTML information needed for presenting the view form of the *project* table.

7.1.3 Controller

The control layer in SHE is also all written as Python modules. For example for creation and adding *project* objects into database as a middle layer between *template* and *model* following code is used (this code is in the module *view.py* in the project and evaluation component or *HE3* app):

```
1. class ProjectCreate(CreateView):
2.     model = Project
3.     fields = ['name', 'link', 'setOfHeuristics', 'description',
                                 'deadline', 'evaluators']
4.     success_url = reverse_lazy('profiles:dashboard:user-dashboard')

5.     def form_valid(self, form):
6.         user = self.request.user
7.         form.instance.manager = user
8.         return super(ProjectCreate, self).form_valid(form)
```

The class *ProjectCreate* defines which fields from table *project* should be presented in *template* to be filled with the user data (line 2,3). It also adds the missing input data (e.g. who created the project object) and saves the object in the database (lines 5-8). Validating the input data is done in the superclass *CreateView*.

For each class in model, there are similar codes in three layers of MVC to create, update and manipulate or delete data. Django forces programmer to follow its structure and this makes codes in applications like SHE, concise, more secure, more readable and easier to maintain and to reuse.

7.2 Merging Evaluations

The manager of the project can merge submitted evaluations and create *merge evaluations*. The *merge and report* component is responsible for the merge and finally producing reports. Image 8.17 shows a *merge desktop*, a view which user deals with during merging phase. In this view, all evaluations which are not merged are categorized based on their evaluators. The manager can click on an evaluation and see its detail in the pop up mode window (see image 8.10). By clicking more than one checkbox a merge button appears on the top of the page. By clicking this button, all the selected evaluations are merged and an update view opens in a new tap and the manager can edit the resulted merged evaluation.

The ids of the selected evaluations are using *ajax* call, dynamically send to the server with this JavaScript function:

```
//Merging evaluations in merge.html

function merge(project_id){
       // getting all the checked ids
    ids = getValueUsingClass('.common-action')
  url='/merge/project/'+ project_id + '/merge_selected_evaluations';
    $.post(url ,
        { csrfmiddlewaretoken: getCookie('csrftoken'), ids:ids },
        function (data) {
            if(data.status == 1){
                open(data.url);
            };
        });
};
```

In the logic layer, the following function is responsible for processing the *ajax* call and merging the evaluations and producing a merge evaluation (this function is in the module *view.py* in the *merge* app from the merge and report component):

```
def mergeEvaluations(request , project_id):
    project = Project.objects.get(pk=project_id)
    resultEval = Evaluation(ofProject =project ,  evaluator=
                            project.manager , merged= True)

    if request.is_ajax():
        ids = request.POST.getlist('ids[]')

        if ids and len(ids) > 1:
            evals = Evaluation.objects.filter(pk__in=ids)
```

```
8.              fields , heurPrincips = mergeFields(evals)
9.              resultEval.__dict__.update(fields)
10.             resultEval.save()
11.             resultEval.heurPrincip.add(*heurPrincips)
12.             mergeScreenshots(resultEval,evals)
13.
14.         # Adding evaluators and evaluations id to the merged evaluation
15.             eval_ids = evals.values_list('pk' ,flat=True)
16.             evaluator_ids = evals.values_list(
                        'evaluator_id' , flat=True).distinct()
17.             resultEval.mergedFromEvaluations.add(*eval_ids)
18.             resultEval.merdedFromEvaluators.add(*evaluator_ids)
19.
20.             url = '/merge/project/'+
                      str(resultEval.id) +'/update-merged-evaluation/'
21.             response = {'status' : 1 , 'url' : url}
                return HttpResponse(json.dumps(response),
                        content_type='application/json')
22.     return HttpResponse('No Success. Select more than one evaluation!')
```

The *mergeEvaluation* function gets the ajax call and checks if more than one evaluation *id* is sent and after that, creates a new merged evaluation and adds all the fields and screenshots of the received evaluations in the resulted merge evaluation. This function also calls *mergeField* function which is responsible for calculating an average of the fields *frequency* and *severity* in the received evaluation in addition to combining their content and returning them to the *mergeEvaluation* function:

```
1.  def mergeFields(evalList):
2.      result ={}
3.      stringList = ['description' , 'recommendation']
4.      stringWithKomma = ['place' ]
5.      forAvg = ['frequency' , 'severity']
6.      for field in stringList :
7.          allvalues = set([ i.__dict__[field] for i in evalList])
8.          result [field] = "\n" .join(allvalues)
9.      for field in stringWithKomma :
10.         allvalues = set([i.__dict__[field]for i in evalList])
11.         result[field] = ",".join(allvalues)
12.     for field in forAvg :
13.         allvalues = [int(i.__dict__[field]) for i in evalList]
14.         result[field] = str(sum(allvalues) / len(allvalues))
15.
16.     heurPrincips =[]
17.     for e in evalList:
18.         heurPrincips.extend(list(e.heurPrincip.all()))
19.     heurPrincips = list(set(heurPrincips))

20.     return result , heurPrincips
```

The resulted merged evaluation is opened in a new page and manager can/should edit and summarize the fields and delete extra information.

7.3 Recommendation Engine

The merge and report component, besides merging evaluations, has the responsibility of recommending similar evaluations for selected evaluations.

7.3.1 Content-based Recommendation

The content-based recommendation is based on the *Nearest Neighbor (NN)* algorithm introduced in the section 5.2.3. For the implementation of this recommendation engine, first step is to build a flat database from all the evaluations in the project. The Function *getEvalSF* retrieves data from relational database and makes a flat database based on *SFrames*, the "scalable tabular and graph data-structures built for out-of-core data analysis and machine learning". (SFrame)

```
1.  def getEvalSF(project):

2.      evaldic = [e.__dict__ for e in project.
                                evaluation_for_project.all()]
3.      evalsf = gl.SFrame()
4.      evalsf['id'] = [str(i['id']) for i in evaldic]
5.      evalsf['title'] = [i['title'] for i in evaldic]
6.      evalsf['place'] = [i['place'] for i in evaldic]
7.      evalsf['description'] = [i['description'] for i in evaldic]
8.      evalsf['recommendation'] = [i['recommendation'] for i in evaldic]
9.      evalsf['tags'] = [i['tags'] for i in evaldic]

10.     return evalsf
```

The second step is to add new column with *TFIDF* score representation for the contents of each evaluation:

```
1.  def evaltfidf(project):
2.
3.      evalsf = getEvalSF(project)
4.      evalsf['desWordCount'] = gl.text_analytics.
                          count_words(evalsf['description'])
5.      evalsf['recommendation'] = gl.text_analytics.
                          count_words(evalsf['recommendation'])
6.      evalsf['tagsWordCount'] =   gl.text_analytics.
                         count_words(evalsf['tags'])
7.      tfidfDes = gl.text_analytics.tf_idf(evalsf['desWordCount'])
8.      tfidfRec = gl.text_analytics.tf_idf(evalsf['recommendation'])
9.      tfidfTags = gl.text_analytics.tf_idf(evalsf['tagsWordCount'])
10.     evalsf['tfidfDes'] = tfidfDes
11.     evalsf['tfidfRec'] = tfidfRec
12.     evalsf['tfidfTags'] = tfidfTags
13.
14.     return evalsf
```

Finally, based on *tfidf* columns and using the *NN* algorithm, similar evaluations are found:

```
1.  def nearestN(eval):
2.      evalsf = evaltfidf(eval.ofProject)
3.      evalrow = evalsf[evalsf['id'] == str(eval.id)]
4.      modelDes = gl.nearest_neighbors.
            create(evalsf, features=['tfidfDes'], label='id')
5.      modelRec = gl.nearest_neighbors.
            create(evalsf, features=['tfidfRec'], label='id')
6.      modelTags = gl.nearest_neighbors.
            create(evalsf, features=['tfidfTags'], label='id')
7.      resultDes = modelDes.query(evalrow).
            filter_by(str(eval.id) , 'reference_label' , exclude= True)
8.      resultRec = modelRec.query(evalrow).
            filter_by(str(eval.id) , 'reference_label' , exclude= True)
9.      resultTags = modelTags.query(evalrow).
            filter_by(str(eval.id) , 'reference_label' , exclude= True)
10.     resultDes = resultDes[resultDes['distance']
                    <= 0.6]['reference_label']
11.     resultRec = resultRec[resultRec['distance']
12.                  <= 0.6]['reference_label']
13.     resultTags = resultTags[resultTags['distance']
14.                  <= 0.6]['reference_label']

15.     result = {'resultDes' : resultDes ,
16.               'resultRec' : resultRec ,
17.               'resultTags': resultTags,}
18.
19.     return result
```

Distinct evaluations in the final result are chosen and return to the view in the following line:

```
allRecommendedEvals = (resultDes | resultRec | resultTags).distinct()
```

7.3.2 Place-based Recommendation

This recommendation is based on two fields in each evaluation: *Place* and *link*. The following function gets the *ajax* call with a selected evaluation (*eval_id*) and returns evaluations with the similar *place* fields:

```
# Module recommend
This function gets an evaluation and return evaluation with
 similar place field
1.  import graphlab as gl
2.  def placebase(eval):
3.      project = eval.ofProject
4.      sf= getEvalSF(project)
5.
6.  # Building search model using graphlab toolkits
7.      placeSearchModel = gl.toolkits._internal.search.
                                create(sf , features= ['id','place'])
8.      result = placeSearchModel.query(eval.place).
                            filter_by(str(eval.id) , 'id' , exclude= True)
9.      return  list(result['id'])
```

The results are later combined with the evaluations with the same link:

```
if eval.link:
        sameLink = allEvaluations.filter(link__exact=eval.link)
        recPlaceBase = (recPlaceBase | sameLink).distinct()
```

7.3.3 Making a Report

In SHE report is a combination of project information, evaluators and their environment information and the merged evaluations. It also contains all the submitted evaluations and heuristic principles used in the project. The merge and report app creates reports using such a function:

```
1.  def reportHtml(request , project_id):
2.     project = Project.objects.get(pk=project_id)
3.     mergedEvals = project.evaluation_for_project.filter(merged = True)
4.     evals = project.evaluation_for_project.filter(merged = False)
5.     environments = Environment.objects.
           filter(creator__in=
           project.evaluators.all()).order_by('creator')

6.     context = { 'project' : project ,
                   'evaluations' : evals.order_by('positivity' ,
                                          'evaluator' , 'severity'),
7.                 'mergedEvals' : mergedEvals,
8.                 'environments' : environments,
9.                 'pos_merge' : mergedEvals.filter(positivity = 'p'),
10.                'neg_merge' : mergedEvals.filter(positivity = 'n'),
11.    }

12.    return   render   (request   ,   template_name='merge/report.html'   ,
    context=context)
```

7.4 Testing

For testing the general performance of SHE in a heuristic evaluation project a test project is conducted using SHE. In this project three different evaluators participated in the evaluation of user interfaces of SHE using SHE itself and finally their evaluations are merged in a report which can be seen in in the appendix B.

As a separate unit test, the recommendation module is also tested with a dataset of more than 50.000 entries in Wikipedia to find similar entries for each person, based on the text analysis of their articles. The image 7.1 shows two examples of the findings for similar persons for the given names *Angelina Jolie* and *Barack Obama* based on their entries in the Wikipedia.

query_label	reference_label	distance	rank
0	Angelina Jolie	0.0	1
0	Brad Pitt	0.784023668639	2
0	Julianne Moore	0.795857988166	3
0	Billy Bob Thornton	0.803069053708	4
0	George Clooney	0.8046875	5

query_label	reference_label	distance	rank
0	Barack Obama	0.0	1
0	Joe Biden	0.794117647059	2
0	Joe Lieberman	0.794685990338	3
0	Kelly Ayotte	0.811989100817	4
0	Bill Clinton	0.813852813853	5

Image 7.1: Similar entries in Wikipedia for Angelina Jolie and Barack Obama

7.5 Summary

In this chapter some code examples are presented to show how different layers of MVC in SHE are implemented. Beside the template engine language in the front end which is similar to the Python syntax and some JavaScript code for dynamic behavior of the front end, all the other codes in SHE are pure Python. The recommendation engine of SHE is implemented in the logic layer of the merge and report component using the flat datasets in *SFrames* and also the implementation of search model and nearest neighbor algorithm in Graphlab. The next chapter gives a walkthrough in SHE to see how a user can use it for HE as an evaluator or a manager.

8 SHE walkthrough

This chapter presents a simple guide to the different parts of SHE *(www.smarthe.net)* and has three parts: how to be a user (to open and use an account), how to be an evaluator and finally how to be a manager of a project in SHE.

8.1 Being a user in SHE

This section goes through some aspects of account managements in SHE, e.g. signing up/in, profile management, password recovery etc. Each functionality is described with the relevant screenshots which a user faces in SHE.

8.1.1 Signing up

In order to use SHE having an account is necessary. Registering an account is done in a simple step with a name which will be used as the username and also an email (see image 8.1):

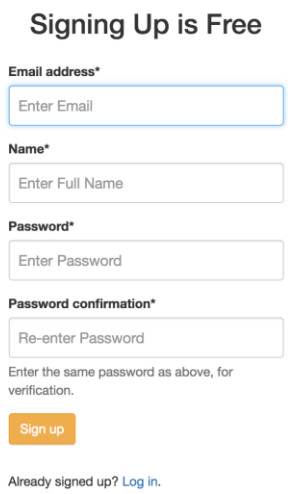


Image 8.1: View for singing up

Name and password can be changed at any time after registration (see image 8.2):

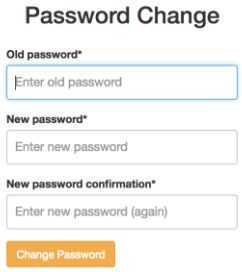


Image 8.2: View for changing a password

8.1.2 Signing In

A registered user should log in to be able to use SHE (see image 8.3):

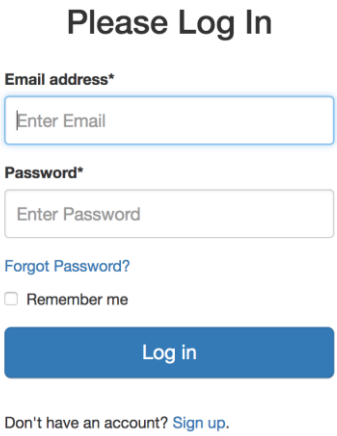


Image 8.3: Log in view

8.1.3 Password Recovery

If users forget their passwords it is possible for them to recover their password using their registered email (see image 8.4):

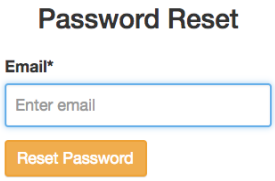


Image 8.4: View for resetting password
using the registered email

8.1.4 Add and Edit Profile

Users can have an editable profile with a photo and a short bio (see image 8.5):

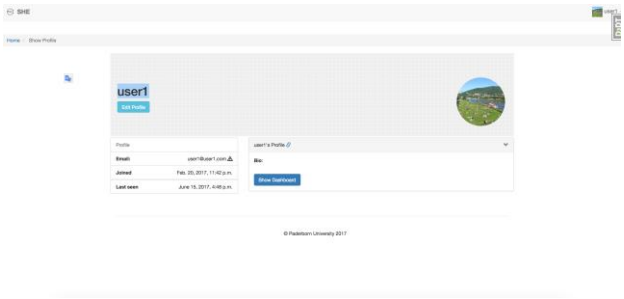


Image 8.5: Profile view of a user

8.1.5 Dashboard

Once a user logged in, he/she sees the dashboard. For the first time dashboard is empty and will be look like this (see image 8.6):

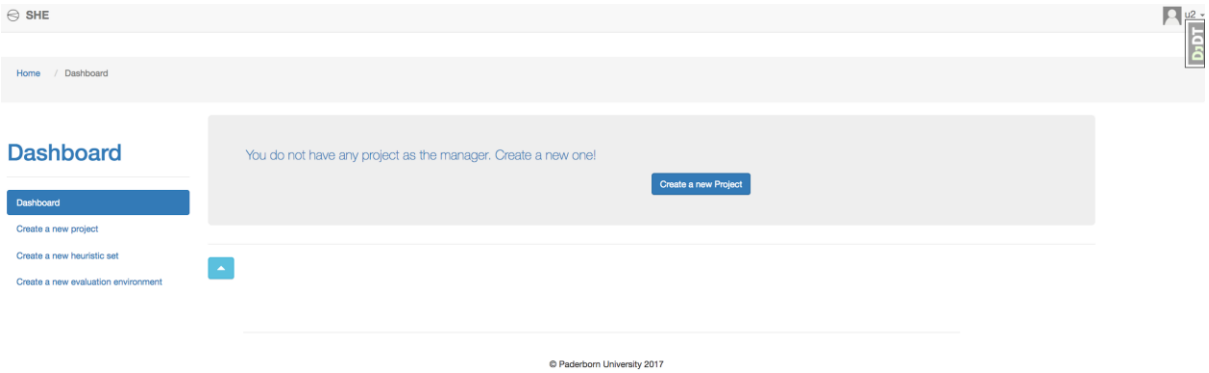


Image 8.6: An empty dashboard is what a user for the first time sees.

Once a user creates his own projects as the manager or becomes an evaluator in the projects of the other users or creates some environment specifics and define new heuristic sets, then the dashboard will be look like this (see image 8.7):

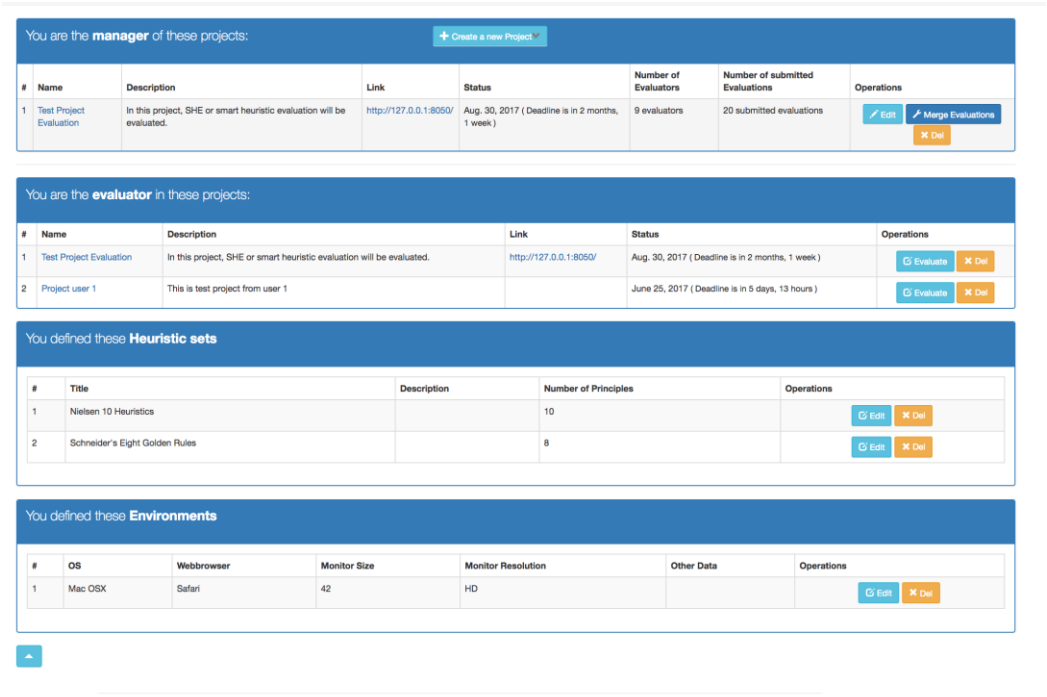


Image 8.7: A dashboard with an overview of all the projects and user defined sets and environments

8.1.6 Project Detail

A user can see the details of each project just by clicking on the title of the project and depending on being a manager or an evaluator, he/she can see different information and operations for that project (see image 8.8).

8.2 Being an Evaluator in SHE

Users will see in their dashboard whether or not they are an evaluator in a project of other users:

If the deadline of a project has not passed, an evaluator can see the blue button *Evaluate*. By clicking this button an evaluator can start adding his/her evaluations to the project. Evaluators can remove themselves as an evaluator in a specific project by pressing the *Del* button. The delete process will be finished after the confirmation in a pop up warning window.

8.2.1 Project Detail for an Evaluator

An evaluator will see the project detail and his/her submitted evaluations (if any) in a view similar to the image 8.8. If the deadline for the evaluation has not passed he/she can add more evaluations to the project.

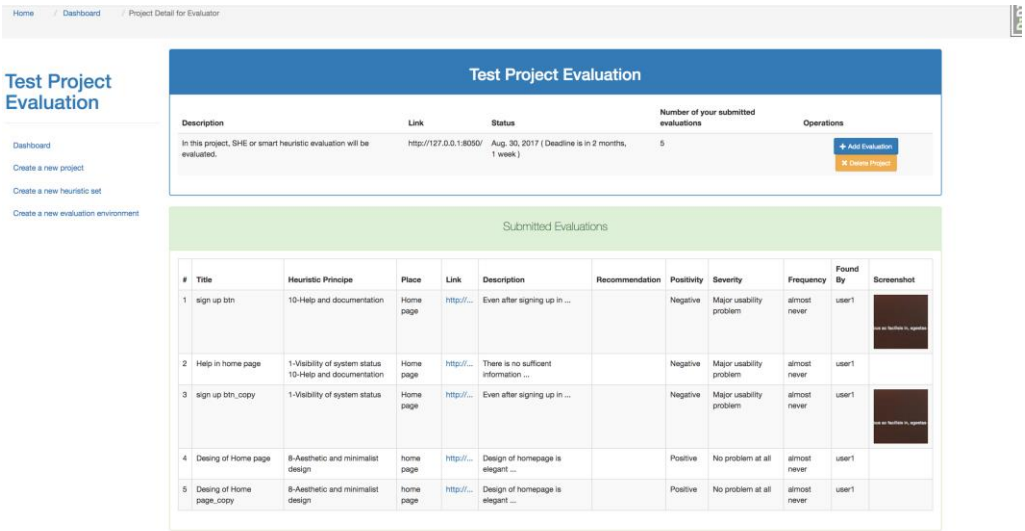


Image 8.8: project detail for an evaluator

8.2.2 Adding a New Evaluation

If the project deadline for the evaluation has not passed an evaluator can press the *add evaluation* button to open an evaluation form similar to image 8.9. By filling the form and pressing the *submit* button at the bottom of the page the evaluation will be saved. In this form some fields are optional and some are necessary. *Title*, *place* and *description* fields are required to be filled. The rest is optional but recommended; e.g. the *link* field in the evaluation of a website page which should be filled in this form: *http://www.example.com*.

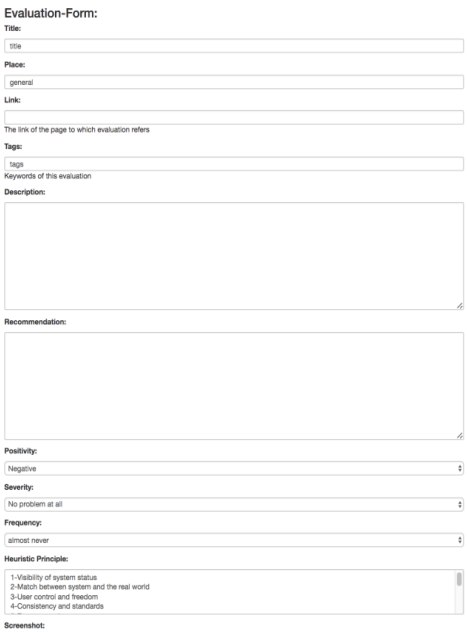


Image 8.9: Adding new evaluation form

8.2.3 See and Edit Evaluation Detail

By clicking on the title of an evaluation in the project detail page the detail of that evaluation will be shown in a pop up window similar to image 8.10. By clicking on the *edit* button a form similar to the image 8.9 opens and the evaluation can be edited again and again.

8.2.4 Add an Environment

Evaluators can register the specifics of their environment so the manager can see the circumstances of the evaluations, e.g. monitor resolution, operation system etc. Creating an environment item is done by clicking on the link *create an evaluation environment* which is in the sidebar of the dashboard and will open a form similar to the image 8.11. An evaluator can create multiple environments.

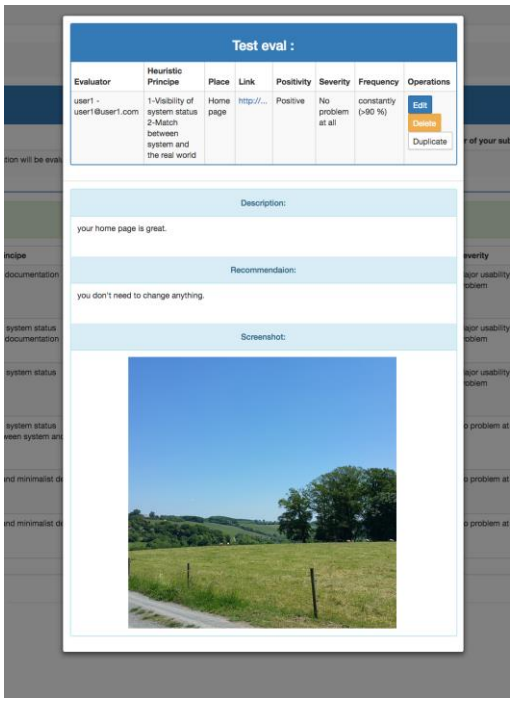


Image 8.10: Evaluation detail in a pop up window

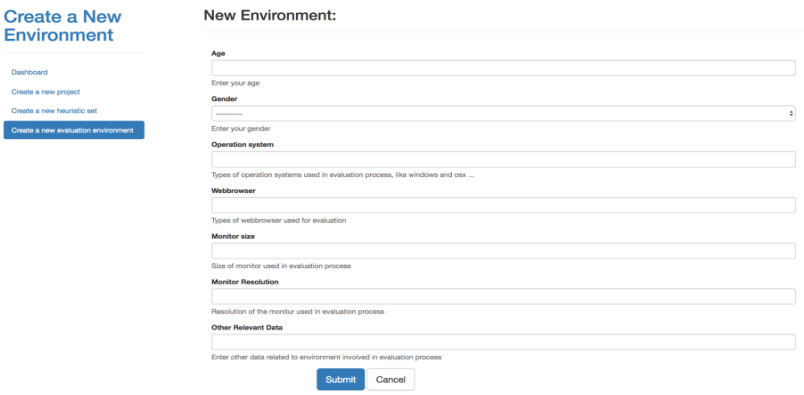


Image 8.11: Creating a new evaluation environment

8.3 Being a Manager in SHE

Being a manager means to define and edit a project. A manager can add and delete other users as evaluators of the project and finally can merge the evaluations and create reports of the results of the evaluation. A manager can define and edit heuristic sets upon which a project is defined.

8.3.1 Defining a Heuristic Set

SHE already has some widely used sets of heuristics like Nielsen heuristics, but a managers can define their own custom set of heuristic sets by clicking on the *create a new heuristic set* tab in the sidebar. A page similar to the image 8.12 will open and after entering a name and a description the set will be created after submitting and a page similar to the image 8.13 will be opened. The set still does not have any principle and therefore is empty. Adding new principles is done by clicking *add new heuristic principle* button (image 8.14).

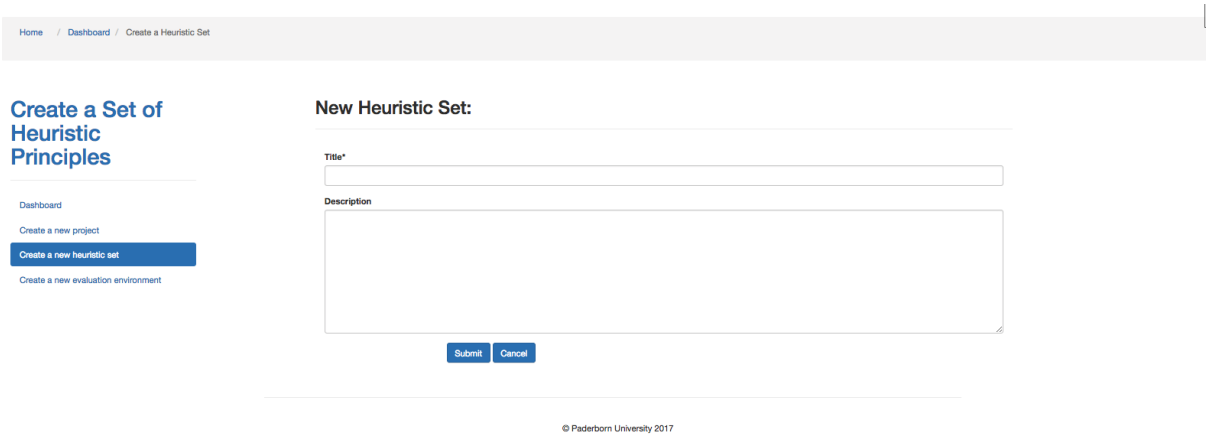


Image 8.12: Defining a new heuristic set by entering a name and a description for the set

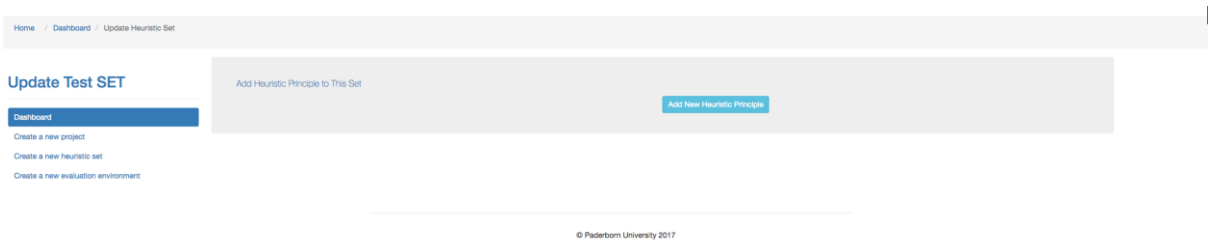


Image 8.13: Empty page of a newly created heuristic set

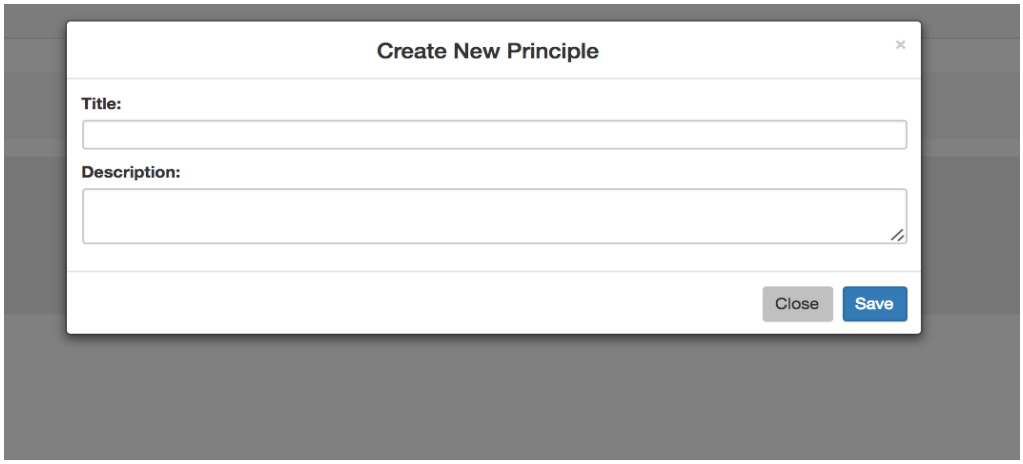


Image 8.14: Create new principle and add them to your set

After adding some principles the set will be similar to the image 8.15. Each principle can be edited or even deleted using operation buttons in the last columns.

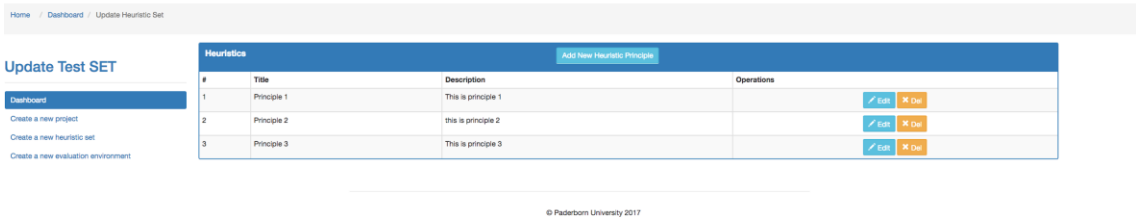


Image 8.15: After adding some principle your set will look like this

NOTE: A set of heuristics which is used in a project cannot be deleted as long as that project is using the set. To delete this set the manager can edit that project and chose another set of heuristics or even delete the whole project to make the set free to be deleted.

8.3.2 Defining a Project

Defining a new project is done by clicking on the *New Project* link in the sidebar of the dashboard to see a form like image 8.16. Except the filed *link*, the other fields are required to be filled. After filling the form, clicking submit will create the new project.

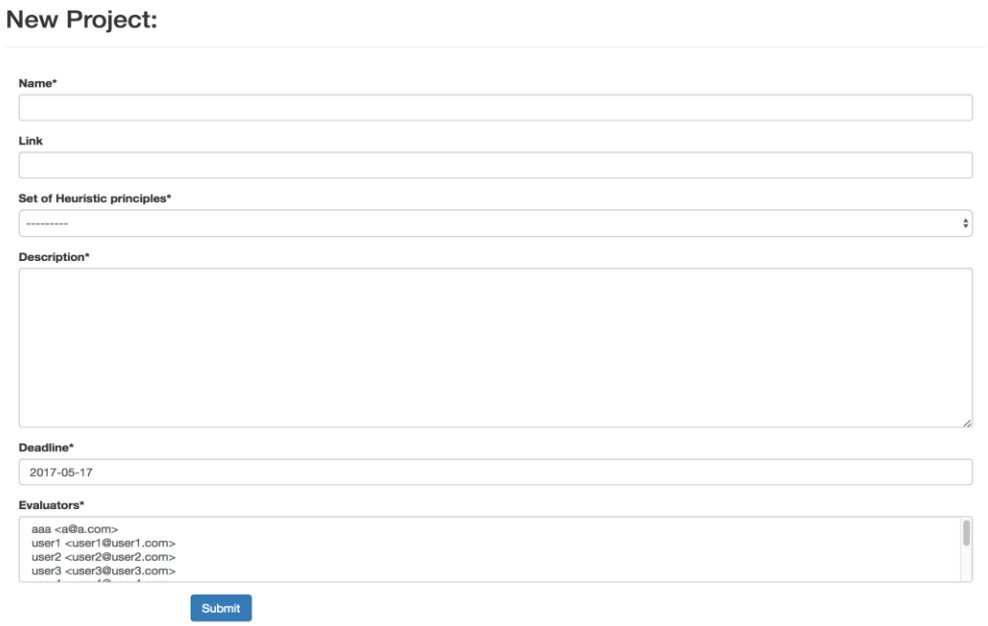


Image 8.16: Form for defining a new project

8.3.3 Merging Evaluations

Manager can merge existing evaluations and create a merged evaluation. The merging can be done anytime but it is better to be after the deadline of the project. After clicking on the *Merge Evaluations* button in the project detail or in the overview of the project in the dashboard merge, the desktop opens a similar page to the image 8.17. In the first column with the name *Evaluations (not merged)* you can see all the evaluations which are not already merged, categorized by the name of their evaluators. By clicking on each evaluation you can see the details of that evaluation like in the image 8.10. Getting similar evaluations is done after clicking the checkbox of any evaluation. Each time a checkbox is checked, place-based and content-based recommendation for merge appear in the middle columns (see image 8.19).

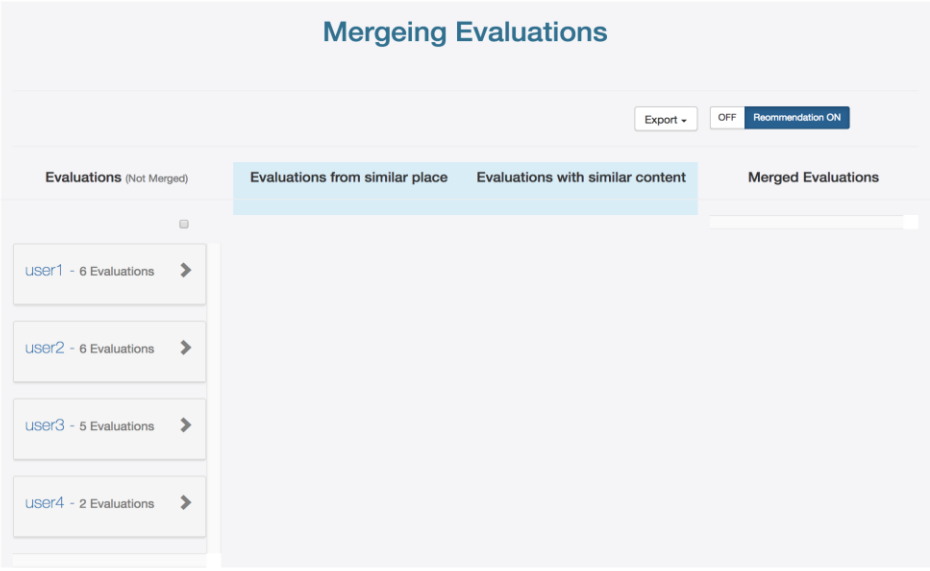


Image 8.17: Merge desktop in SHE

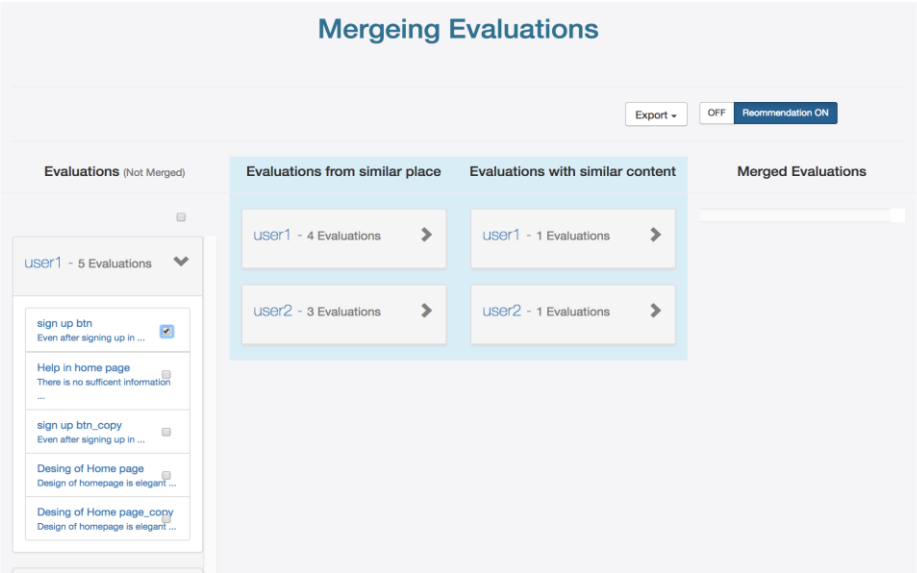


Image 8.18: After clicking any evaluation panel in the leftmost column similar evaluation are shown in the two middle columns

By choosing more than two checkboxes a blue button with the name *Merge Selected Evaluations* appears on the top left of the page (image 8.19). Clicking this button will combine the information of all the selected evaluations to create a new merged evaluation. An evaluation form page with the information of the new merged evaluation will be opened which can/should be edited to summarize the combined data in its fields.

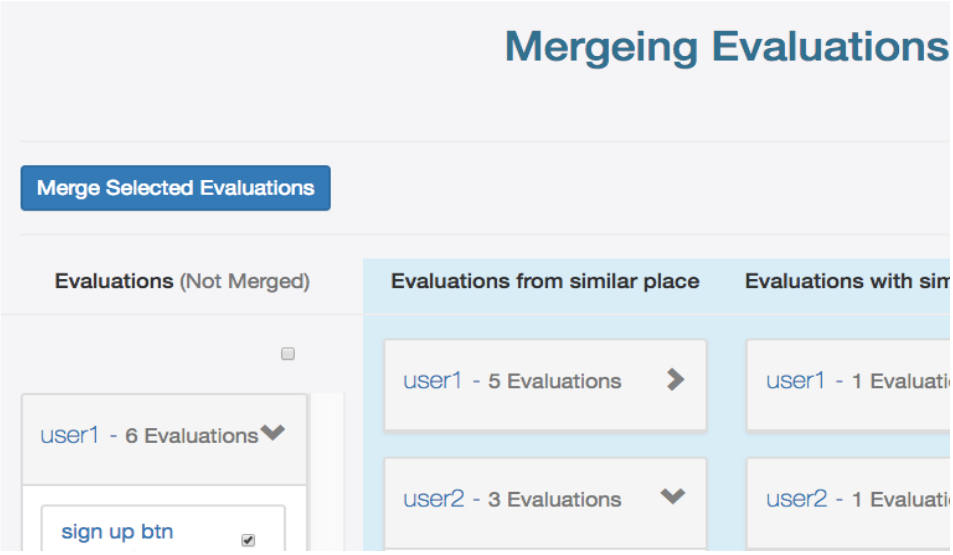


Image 8.19: For merging selected evaluations the button *merge selected evaluations* should be clicked

The new merged evaluations appear in the last column on the left side of the screen (see image 8.20):

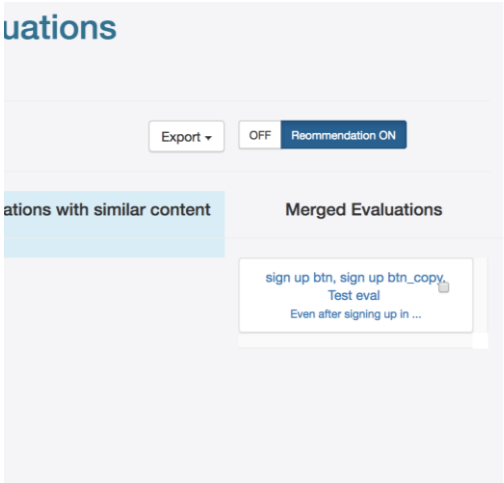


Image 8.20: New merged evaluations appear
In the rightmost column in merge desktop

8.3.4 Exporting Evaluations

In project detail page a manger can find different *Export* buttons which give different export options in a dropdown menu to export both merged evaluations and normal evaluations separately (see image 8.21):

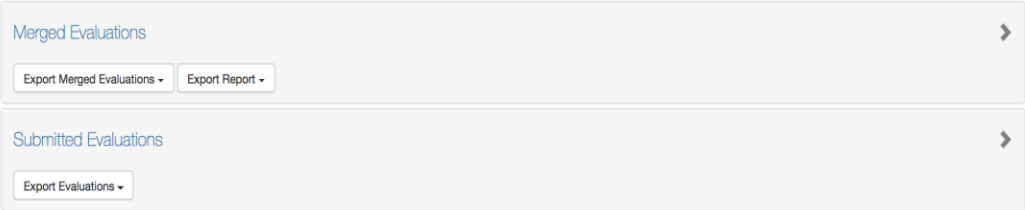


Image 8.21: Export buttons on the evaluation panels can be used to export evaluations or a report in a different format

8.3.5 Making a Report

The manager can find can find *Export* button for producing and exporting a report in the project detail page and also on the top of the merge desktop page (see image 8.21).

8.4 Summary and Future Works

The three main goals of SHE are to be, as a collaborative web application for HE, a) a multi user online application, b) with the possibility of logging evaluations for a project based on a custom set of heuristics and c) with the smart support of the merging of the evaluations for the end report through finding the similarities between the logged evaluations and recommending them as the merge candidates. This chapter showed how SHE fulfills the mentioned goals and therefore overcomes the shortcomings of other existing tools.

Although SHE provides sufficient supports for the main procedures in a HE project but still there is always room for improvements. This is a list of changes or features which can be added in the future versions of SHE:

- The front end of SHE has plenty of room for improvements. The user interfaces of SHE just gets the job done but certainly, they need to be redesigned and polished to be more user friendly and efficient.
- The recommendation engine in SHE can be improved to recognize similarities between screenshots. This engine can also have extra options for recommendation, e.g. based on heuristics.
- The merged evaluation resulted in combining different evaluations has always extra information and needs to be edited by the manager. This process can be improved by adding interactive interface features so the manager can participate in choosing which information should be in the resulted merged evaluation and which one should be excluded.
- The merge desktop can have more categorizing and ordering features based on different data in each evaluation, e.g. heuristics, severity, frequency etc.
- The exported report from SHE has only the project information in a raw form and as a result, the report needs to be reedited in other editors. Report producing in SHE can be more interactive and also SHE can embed rich text editors to produce end reports so there is no need of reediting the report in other tools by the manager.

Bibliography

Abbasifard, M., Ghahremani, B., & Naderi, H. (2014, June). A Survey on Nearest Neighbor Search Methods. *International Journal of Computer Applications (0975 – 8887) , 95*(25), pp. 39-52.

Andrews, K. (2005). *Skeleton Heuristic Evaluation Report.* Retrieved July 2017, from http://courses.iicm.edu/hci/practicals/materials/en/he/he.html.

Andrews, K. (2017). *Lecture Notes: Human-Computer Interaction.* Retrieved from Graz University of Technology,: http://courses.iicm.tugraz.at/hci/

Bias, R. (1991, September). Walkthroughs: Efficient collaborative testing. *IEEE Software 8,5*, pp. 94-95.

Bootstrap . (n.d.). Retrieved April 10, 2017, from www.getbootstrap.com

Data mining. (n.d.). Retrieved July 5, 2017, from Define Data mining at Dictionary.com: http://www.dictionary.com/browse/data-mining

Django docs. (n.d.). Retrieved Mars 4, 2017, from https://docs.djangoproject.com/en/1.11/

Django Project. (n.d.). Retrieved Mars 4, 2017, from www.djangoproject.com

Edge2.2. (n.d.). *Django Edge Documentation*. Retrieved June 19, 2017, from https://django-edge.readthedocs.io/

Gallello, C. (2014, Dec 20). *Low cost usability testing.* Retrieved April 8, 2017, from Medium: https://medium.com/@cgallello/low-cost-usability-testing-61b5f8a2a1be

Gallello, C. (n.d.). *UX-Check*. Retrieved April 8, 2017, from http://www.uxcheck.co/

Graphlab Create. (n.d.). Retrieved Mars 4, 2017, from Turi: www.turi.com

Instone, K. (1997). *Site Usability Heuristics for the Web.* Retrieved July 2017, from http://www.instone.org/heuristics

ISO 13407. (1999). *Human-Centred Design Processes for Interactive Systems*. Retrieved from http://www.iso.org/

ISO 9241-11. (1998). *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs).* Retrieved from Part 11: Guidance on Usability: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en

Jinja2. (n.d.). Retrieved June 18, 2017, from http://jinja.pocoo.org/docs/latest/

jQuery. (n.d.). Retrieved April 10, 2017, from www.jquery.com

Kahn, M., & Prail, A. (1994). Formal usability inspections. In *Usability Inspection Methods* (Nielsen, J.; Mack, R.L. ed.).

Lewis, C., Polson, P., Wharton, C., & Rieman, J. (1992). Cognitive walkthroughs: A method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies 36,5*, pp. 741–773.

Libscore. (n.d.). Retrieved April 10, 2017, from www.libscore.com

Loitzl, M. (2006). *The Heuristic Evaluation Manager (HEM): An Online Collaborative Environment for Heuristic Evaluation.* Retrieved from Diss. Graz U of Technology: Http://www.iicm.tugraz.at/thesis/mloitzl.pdf

Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., & Hellerstein, J. (2012, April). Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment VLDB, 5*(8), pp. 716-727.

Lutz, M. (2013). *Learning Python* (5 ed.). Sebastopol, CA, USA: O'Reilly Media.

Muller, M. J., Matheson, L., Page, C., & Gallup, R. (1998). Methods & Tools: Participatory Heuristic Evaluation. Interactions. *Interactions, 5*(5), pp. 13-18.

Nielsen, J. (1992). Finding usability problems through heuristic evaluation. *Proc. ACM CHI' 92 Conf.*, (pp. 373-380). Monterey, CA.

Nielsen, J. (1993). *Usability Engineering.* Boston: Academic.

Nielsen, J. (1994). Enhancing the explanatory power of usability heuristics. *Proc. CHI '92 Conf.* (pp. 152-158). Boston, Massachusetts, USA: ACM New York, NY.

Nielsen, J., & Mack, R. (1994). *Usability Inspection Methods.* New York, NY: John Wiley & Sons.

Nielsen, J., & Molich, R. (1990, April). Heuristic Evaluation of User Interfaces. *Proc. ACMCHI'90 Conf.* (pp. 249-256). Seattle, WA: April.

Pypl popularity of programming language. (2016, June). Retrieved Mars 4, 2017, from http://pypl.github.io/PYPL.html

Python - Web framework rankings. (n.d.). Retrieved April 10, 2017, from HotFrameworks: https://hotframeworks.com/languages/python

Python docs. (n.d.). Retrieved June 2, 2017, from www.python.org/doc

Rajaraman, A., & Ullman, J. D. (2011). *Mining of Massive Datasets.* Cambridge University Press.

Rubin, J. (1984). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests.* John Wiley and Sons, Inc.

SFrame. (n.d.). Retrieved June 2, 2017, from Github: https://github.com/turi-code/SFrame

Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., & Elmqvist, N. (2016). *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Sixth Edition.* Pearson.

Skinner, G., & McMullin, J. (2003). *Usability Heuristics for Rich Internet Applications.* . Retrieved July 2017, from Boxes and Arrows Website: www.boxesandarrows.com/usability-heuristics-for-rich-internet-applications/

The 2015 top ten programming languages. (2015, July). Retrieved Mars 2, 2017, from Spectrum: http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages

UzReview. (n.d.). *mozdev.org*. Retrieved April 9, 2017, from uzilla- heuristicreview: http://uzilla.mozdev.org/heuristicreview.html

Van Rossum, G. (2009, January). Retrieved July 5, 2017, from http://python-history.blogspot.de/2009/01/pythons-design-philosophy.html

Appendix A: Running SHE locally

To use SHE one can access it under the domain name *www.smarthe.net* or to run the source code locally through these steps:

1) Cloning the source code from *gitlab* or *github*:

```
$ git clone https://erahnema@git.cs.upb.de/erahnema/she.git

#Or alternatively from github: $ git clone https://github.com/boogh/she.git
```

2) Creating a graphlab environment using Anaconda Python Environment (see https://turi.com/download/install-graphlab-create-command-line.html):

```
# Create a new conda environment with Python 2.7.x
$ conda create -n gl-env python=2.7 anaconda=4.0.0

# Activate the conda environment
$ activate gl-env

# Ensure pip is updated to the latest version
$ conda update pip

# Install licensed copy of GraphLab Create
$ pip install --upgrade --no-cache-dir https://get.graphlab.com/GraphLab-
Create/2.1/your registered email address here/your product key here/Graph
Lab-Create-License.tar.gz
```

3) Installing the requirements of SHE and finally running SHE:

```
$ pip install -r requirements.txt

$ python manage.py makemigrations

$ python manage.py migrate

$ python manage.py runserver
```

Note: In windows Django should be added to the path.

Appendix B: Result of a Test Project Using SHE

As a general test, SHE is used for a heuristic evaluation project in which three evaluators participated in the evaluation of SHE interfaces using SHE itself. The evaluation was based on Nielsen heuristic and the results were merged based on the place of evaluations:

Evaluation of SHE - Report

Project Description:

SHE or Smart Heuristic Evaluation is a web application for supporting heuristic evaluation. In this project user interfaces of SHE will be evaluated. The evaluation is based on Nielsen heuristics . - http://34.212.137.160/

Evaluators:

These are the evalutors in this project:

- abbas - abbas@abbas.com
- javad - javad@gmail.com
- martin - martin@gmail.com

Evaluator Environments:

#	Evaluator	Age	Gender	Webbrowser	OS	Monitor Size	Monitor Resolution	Other Data
1	abbas <abbas@abbas.com>	27	1	Chrome	Windows 10	17 inch	HD	
2	javad <javad@gmail.com>	29	1	chrome	OSX	13 inch		
3	martin <martin@gmail.com>	26	1	Chrome and Firefox	Windows 10	23	HD	

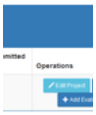
Positive Merged evaluations:

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Severity	Frequency	Found By	Screenshots
1	Positive points	8- Aesthetic and Minimalistic Design	Homepage, general	()	1) In general SHE has minimalist design with only the necessary functions. specially in merge desktop. Although it needs to be have more harmonic design with more aesthetic details. 2) The Homepage has a very nice picture as the background		No problem at all	almost never	abbas martin	

Negative Merged Evaluations:

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Severity	Frequency	Found By	Screensho
1	About page issues	8- Aesthetic and Minimalistic Design 4- Consistency and Standards 7- Flexibility and Efficiency of Use	About page	()	1) The header of About page is very big and has some useless spaces. (Header in about page has plenty of unnecessary spaces around it.) 2) Text in about page should not be center aligned 3) On the bottom of About page, your email "erahnema@mail.uni-paderborn.de" should be clickable. Users might want to open it in outlook or any email programm. 4) The text in the About page should be left justified not center alignment 5) Design of the about page or information about the website and the designer is not that enough and is not attractive to read and look at the page. Specially, the first part, SHE and A Smart Tool for an Smart Heuristic Evaluation is completely annoying for users' eyes. 6) There is a text about the project on the help page. I think anything about the project should come on About page and it shouldn't repeat it on this page.		Cosmetic problem	occasionally (11-50 %)	abbas javad martin	 (/media/sc 08-01/5.jp
2	Dashboard issues	8- Aesthetic and Minimalistic Design 1- Visibility of System Status 3- User Control and Freedom	Dashboard	()	1) Tables in dashboard with long link or description lose their responsive shape 2) There is no way to communicate with the manager of the website or the managers of the projects. Evaluators can not send a message and ask question from the managers. 3) when screen size changes dashboard is not any more organized		No problem at all	rarely (< 10 %)	javad martin	 (/media/sc 08-01/proj

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Severity	Frequency	Found By	Screensho
3	Help page issues	8- Aesthetic and Minimalistic Design 1- Visibility of System Status 4- Consistency and Standards	Help page	http://... (http://34.212.137.160/help)	1) The left menu in the help page should be wider, so the links and the arrows come to the same line. Also, the left menu is almost sticking to the header text box. 2) In mid size screen the help page loses the structure and even a part of title is not readable any more. 3) On this page, there are many titles but no list of them at the top of the page. 4) Some of the images in the Help page are very big compared to the others. For example the login image is very big but the password recovery image is small 5) On the top of any title, there is a useless space.	You should have a table of contents, so users can click on any title and go directly to that part.	Cosmetic problem	regularly(51-89 %)	abbas martin	 (/media/sc 08-01/6.jp  (/media/sc 08-01/7.jp  (/media/sc 08-01/help
4	Home page issues	8- Aesthetic and Minimalistic Design 1- Visibility of System Status 10- Help and Documentation	Homepage	http://... (http://34.212.137.160/)	1) There is a beautiful picture in the Homepage, but I believe it is not related to the subject of the website. Most of the people consider the website as a real estate agency website, when they see the Foto at first glance. 2) Users do not get clear sense of the website. 3) There are a lot of annoying white gaps at the bottom of home page. 4) Design of the home page is not that much attractive to motivate someone who visited your website once to comeback and check it again. 5) Even after signing up in the home page there is a sign up button. 6) There is no sufficient information about SHE in home page. 7) Because of the color of "Smart Heuristic Evaluation", we can read it hard. 8) There is an annoying white gap under the top menu	1)Remove unnecessary gaps 2) Instead of sign up button should be a go to the dashboard button	Cosmetic problem	occasionally (11-50 %)	abbas javad martin	 (/media/sc 07-31/4.jp  (/media/sc 07-31/2.jp  (/media/sc 07-31/3.jp

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Severity	Frequency	Found By	Screensho
5	Project detail issues	8- Aesthetic and Minimalistic Design 1- Visibility of System Status 3- User Control and Freedom 4- Consistency and Standards 7- Flexibility and Efficiency of Use	Project detail	()	1) In project detail there is no functionality of sorting for evaluation tables. After submitting an evaluation it is not clear where it goes in table. 2) There is no possibility for an evaluator to export his own evaluations. 3) The buttons in the project table are changing position in different screen size. Even in large screen they are not organized. 4) The add project page is too wide and I have to scroll vertically	A drop down set of buttons will be a better choice.	Cosmetic problem	rarely (< 10 %)	javad martin	 (/media/sc 08-01/proj
6	General issues	9- Help Users Recognize, Diagnose, and Recover from Errors 2- Match Between System and Real World 4- Consistency and Standards 10- Help and Documentation 7- Flexibility and Efficiency of Use	profile page, evaluation-form, sidebar, Login Page, general	()	1) There is almost no where in SHE possibility of getting help for a local functionality. The only choice is to go the main help. submitting multiple evaluations is done not efficiently because after each submission you go back to the project detail. 2) I think your title in the login page is unnormal. 3) After signing in instead of dashboard user goes to the profile page. 4) In sidebar all the links only represented with text without any icons.	Adding a button for continuing the evaluation without going back to the project detail You can omit the word "please" in login page	Minor usability problem	occasionally (11-50 %)	abbas martin	

All Evaluations:

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Pos
1	White gap between top menu and picture	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	There is an annoying white gap under the top menu		Neg

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Pos
2	Problem in reading the title	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	Because of the color of "Smart Heuristic Evaluation", we can read it hard.	You can put a transparent black box with under it	Neg
3	Login Title	2- Match Between System and Real World	Login Page	http://... (http://34.212.137.160/login/)	I think your title in the login page is unnormal.	You can omit the word "please"	Neg
4	Header useless spaces	8- Aesthetic and Minimalistic Design	About page	http://... (http://34.212.137.160/about/)	The header of About page is very big and has some useless spaces.	You can make it shortened	Neg
5	Email text isn't clickable	7- Flexibility and Efficiency of Use	About page	http://... (http://34.212.137.160/about/)	On the bottom of About page, your email "erahnema@mail.uni-paderborn.de" should be clickable. Users might want to open it in outlook or any email programm.		Neg
6	Big Images in the Help page	8- Aesthetic and Minimalistic Design	Help page	http://... (http://34.212.137.160/help/)	Some of the images in the Help page are very big compared to the others. For example the login image is very big but the password recovery image is small		Neg
7	White gaps again	8- Aesthetic and Minimalistic Design	Help page	http://... (http://34.212.137.160/help/)	On the top of any title, there is a useless space.		Neg
8	Annoying white gaps	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	There are a lot of annoying white gaps at the bottom of home page.	Remove unnecessary gaps	Neg
9	About text center alignment	8- Aesthetic and Minimalistic Design 4- Consistency and Standards	About page	http://... (http://34.212.137.160/about/)	The text in the About page should be left justified not center alignment		Neg

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Pos
10	Left menu in Help page	8- Aesthetic and Minimalistic Design	Help page	http://... (http://34.212.137.160/help/)	The left menu in the help page should be wider, so the links and the arrows come to the same line. Also, the left menu is almost sticking to the header text box.		Neg
11	About SHE in the Help page	4- Consistency and Standards	About page	http://... (http://34.212.137.160/help/)	There is a text about the project on the help page. I think anything about the project should come on About page and it shouldn't repeat it on this page.		Neg
12	Table of contents	4- Consistency and Standards	Help page	http://... (http://34.212.137.160/help/)	On this page, there are many titles but no list of them at the top of the page.	You should have a table of contents, so users can click on any title and go directly to that part.	Neg
13	Picture in the Homepage	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	There is a beautiful picture in the Homepage, but I believe it is not related to the subject of the website. Most of the people consider the website as a real estate agency website, when they see the Foto at first glance.		Neg
14	No connection between mangers and evaluators	1- Visibility of System Status	Dashboard	http://... (http://34.212.137.160/users/me/dashboard/)	There is no way to communicate with the manager of the website or the managers of the projects. Evaluators can not send a message and ask question from the managers.		Neg
15	Design in the add new project page	8- Aesthetic and Minimalistic Design	Add project - Dashboard	http://... (http://34.212.137.160/users/me/dashboard/add_project/)	The add project page is too wide and I have to scroll vertically		Neg
16	Homepage design	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	Design of the home page is not that much attractive to motivate someone who visited your website once to comeback and check it again.		Neg
17	Purpose of website	10- Help and Documentation	Homepage	http://... (http://34.212.137.160/)	Users do not get clear sense of the website.		Neg

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Pos
18	Website design	8- Aesthetic and Minimalistic Design	About	http://... (http://34.212.137.160/about/)	Design of the about page or information about the website and the designer is not that enough and is not attractive to read and look at the page. Specially, the first part, SHE and A Smart Tool for an Smart Heuristic Evaluation is completely annoying for users' eyes.		Neg
19	Help in home page	10- Help and Documentation	home page	http://... (http://34.212.137.160)	There is no sufficient information about SHE in home page.		Neg
20	Not responsive page in small screens	8- Aesthetic and Minimalistic Design 3- User Control and Freedom	dashboard	http://... (http://34.212.137.160/users/me/dashboard/)	when screen size changes dashboard is not any more organized		Neg
21	Design of dashboard	8- Aesthetic and Minimalistic Design	dashboard	http://... (http://34.212.137.160/users/me/dashboard/)	Tables in dashboard with long link or description lose their responsive shape		Neg
22	Design in about page	8- Aesthetic and Minimalistic Design	about page	http://... (http://34.212.137.160/about/)	Text in about page should not be center aligned		Neg
23	Design of header in about page	8- Aesthetic and Minimalistic Design	about page	http://... (http://34.212.137.160/about/)	Header in about page has plenty of unnecessary spaces around it.		Neg
24	Not organized buttons	8- Aesthetic and Minimalistic Design 1- Visibility of System Status 4- Consistency and Standards	Project detail	http://... (http://34.212.137.160/users/me/dashboard/project/2/)	The buttons in the project table are changing position in different screen size. Even in large screen they are not organized.	A drop down set of buttons will be a better choice.	Neg
25	sign up button	1- Visibility of System Status	Home page	http://... (http://34.212.137.160/)	Even after signing up in the home page there is a sign up button.	e.g. Instead of sign up button should be a go to the dashboard button	Neg
26	Profile page instead of dashboard	4- Consistency and Standards 7- Flexibility and Efficiency of Use	profile page	http://... (http://34.212.137.160/users/me)	After signing in instead of dashboard user goes to the profile page.		Neg
27	design in help page	8- Aesthetic and Minimalistic Design 1- Visibility of System Status	help page	http://... (http://34.212.137.160/help/)	In mid size screen the help page loses the structure and even a part of title is not readable any more.		Neg

#	Title	Heuristic Principe	Place	Link	Description	Recommendation	Pos
28	Lack of sorting funcionality	1- Visibility of System Status 3- User Control and Freedom 4- Consistency and Standards	project detail	http://... (http://34.212.137.160/users/me/dashboard/project/2/)	In project detail there is no functionality of sorting for evaluation tables. After submitting an evaluation it is not clear where it goes in table.		Neg
29	Lack of icons in sidebar	9- Help Users Recognize, Diagnose, and Recover from Errors 4- Consistency and Standards	sidebar	()	In sidebar all the links only represented with text without any icons.		Neg
30	Adding evaluations is not efficient	7- Flexibility and Efficiency of Use	evaluation-form	http://... (http://34.212.137.160/users/me/dashboard/project/AddEvaluation/2/)	submitting multiple evaluations is done not efficiently because after each submission you go back to the project detail.	Adding a button for continuing the evaluation without going back to the project detail	Neg
31	Lack of export functionality for evalutor	3- User Control and Freedom 7- Flexibility and Efficiency of Use	general, project detail for evaluator	()	There is no possibility for an evaluator to export his own evaluations		Neg
32	Lack of local help	10- Help and Documentation	general	()	There is almost no where in SHE possibility of getting help for a local functionality. The only choice is to go the main help.		Neg
33	Nice Background Picture	8- Aesthetic and Minimalistic Design	Homepage	http://... (http://34.212.137.160/)	The Homepage has a very nice picture as the background		Pos
34	Minimalistic design	8- Aesthetic and Minimalistic Design	general	()	In general SHE has minimalist design with only the necessary functions. specially in merge desktop. Although it needs to be have more harmonic design with more aesthetic details.		Pos

Heuristic set:

This project used **Nielsen Heuristics** set of heuristcs.

Nielsen Heuristics :

Jakob Nielsen's 10 general principles for interaction design. They are called 'heuristics' because they are broad rules of thumb and not specific usability guidelines.

1. Principle 1 - 1- Visibility of System Status :

   The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

2. Principle 2 - 2- Match Between System and Real World :

   The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. Principle 3 - 3- User Control and Freedom :

   Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. Principle 4 - 4- Consistency and Standards :

   Users should not have to wonder whether different words, situations, or actions mean the same thing.

5. Principle 5 - 5- Error Prevention :

   Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

6. Principle 6 - 6- Recognition Rather than Recall :

   Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. Principle 7 - 7- Flexibility and Efficiency of Use :

   Accelerators (unseen by the novice user) may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

8. Principle 8 - 8- Aesthetic and Minimalistic Design :

   Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

9. Principle 9 - 9- Help Users Recognize, Diagnose, and Recover from Errors :

   Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

10. Principle 10 - 10- Help and Documentation :

    Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.
