Graphical User Interface for Timetabling at TekNat with Standardised Output

Axel Andrén
Abstract

Graphical User Interface for Timetabling at TekNat with Standardised Output

Axel Andrén

Uppsala University’s “Vetenskapsområdet för teknik och naturvetenskap” (Disciplinary Domain of Science and Technology), or just TekNat, has been moving towards automating their scheduling process for several years. This project has been part of that effort, and had the goal of showing that requirements could be gathered for a scheduling interface that satisfied both scheduling staff and teachers, as well as showing that these requirements could then be implemented. Human-Computer Interaction, or HCI, was used to aid in these tasks, as it is a field concerned with how to design effectively for intended end users, and has a variety of techniques and methods to aid in accomplishing this.

Some requirements were known beforehand, such as the interface needing to have a standard format for its output, while most were gathered using HCI methods such as semi-structured interviews and paper prototyping. The final requirements specified what was desired for the interface to improve on the current system and be faster, easier, more enticing to use. A functional GUI was implemented that followed these requirements, with some reservations, and improved after feedback from three rounds of testing with the staff.
Contents

1 Introduction ............................................. 8
   1.1 Background ......................................... 8
   1.1.1 Human–Computer Interaction .................. 9
   1.2 Objective ........................................... 10
   1.3 Limitations ......................................... 10
   1.4 Section contents ..................................... 11

2 Previous work ......................................... 12
   2.1 Overview ........................................... 12
   2.2 Similar applications and products ............... 13
      2.2.1 innoCampus' MosesKonto .................... 13
      2.2.2 Wise Timetable ................................ 16
      2.2.3 Cobi ........................................... 18
      2.2.4 Lantiv Scheduling Studio ..................... 19
   2.3 Research and reports ................................ 21
      2.3.1 On the Design and Development of User Interfaces in Interactive Scheduling Systems ............. 21
      2.3.2 Display Requirements for an Interactive Rail Scheduling Display .............................. 22
      2.3.3 Scheduling Home Control Devices .......... 23
   2.4 Conclusion .......................................... 25

3 Theory .................................................. 26
   3.1 Human–Computer Interaction ...................... 26
   3.2 Requirements gathering ............................ 27
   3.3 User Analysis ....................................... 28
   3.4 Prototyping .......................................... 29
   3.5 Visual design ....................................... 29
   3.6 Conclusion .......................................... 30

4 Method .................................................. 32
   4.1 Contextual Design ................................... 32
   4.2 Requirements gathering methods ................. 33
      4.2.1 Interviews ...................................... 34
      4.2.2 PACT; People, Activities, Context, Technologies .................................................. 34
      4.2.3 User personas ................................... 35
      4.2.4 Other methods ................................... 36

5
### 4.3 User Analysis methods

**4.3.1 Access, Usability, Acceptability and Engagement**

### 4.4 Prototyping

**4.4.1 Paper prototyping**

### 4.5 Visual design

### 5 Results

**5.1 Overview**

**5.2 Requirements gathering**

**5.3 Prototypes**

**5.4 Final result**

**5.5 Conclusion**

### 6 Evaluation

**6.1 Language choice for making the GUI**

**6.1.1 Results of choosing Java applets**

**6.2 Accessibility, Usability, Acceptability, Engagement**

**6.3 Interviews**

**6.4 Prototype**

**6.4.1 Paper prototype**

**6.4.2 Digitalization attempt 1**

**6.4.3 Paper prototype 2**

**6.4.4 Digitalization attempt 2**

**6.4.5 Proof of concept**

**6.4.6 Prototype 1**

**6.4.7 Prototype 2**

**6.4.8 Unfinished future prototype**

**6.5 Testing methods**

**6.6 Personal challenges**

### 7 Discussion

**7.1 Future work**

**7.1.1 Specific features**
Chapter 1

Introduction

This project intends to show it is possible to gather requirements of an interface for the staff at Uppsala University's "Vetenskapsområdet för teknik och naturvetenskap" (Disciplinary Domain of Science and Technology), or just TekNat, to make schedule requests in, and then show how such requirements could be implemented.

1.1 Background

Currently, the teaching staff at Uppsala University’s Vetenskapsområdet för teknik och naturvetenskap (Disciplinary Domain of Science and Technology), or TekNat, submit their timetable preferences for each semester directly to the scheduling staff by writing down their requests on paper and submitting those, or going over to the scheduling staff to talk in person. If they fail to submit anything, the scheduling staff will try to contact them directly, but if this also fails the staff are forced to use last years schedule for that person or outright guess at what will work for the person in question. Even after the data is gathered, it still takes several months to create the timetables, which is not only a lot of work but also means the teachers have to submit their preferences far in advance.

Yet despite all this effort, there are still collisions in the timetables. As such, a way to ease or outright automate the process has been sought, and there has been some work done towards that end. There was a feasibility study done by Johan Ludde Lundin [Lundin2011] in 2011, which determined that it was indeed possible, while Henning Hellkvist and William Sjöstedt [HellkvistSjoestedt2012] successfully made an automated timetable generator in 2012. However, the focus of their work was on creating the logic core behind the automated process, rather than on producing something that the staff would consider acceptable to use. As such, the interface for this logic core ended up being technical and difficult to use — see fig [1]. A more easily used interface to that logic core was desired, and this project sought to determine what exactly was needed of such an interface. The resulting requirements could then be implemented in a graphical user interface.

Even if the scheduling process cannot be completely automated with this interface, the ability for teachers to quickly submit their preferences in a common format should be an improvement over the current process and make it considerably easier for the scheduling staff to keep track of all the required data. The standardised output is also expected to be easier to translate such that it works with Hellkvist and Sjöstedt’s program, but this project did not focus on direct compatibility with it.


1.1.1 Human–Computer Interaction

When designing interfaces or functionality, even for physical products, it can be difficult to produce something that, people understand how to use, are interested in using, and is sufficiently powerful for the intended task. For designs involving computers specifically, we have an entire field of study, Human–Computer Interaction, which is concerned with how to make designs attractive not just in looks, but in how they are used.

The difference between a design that merely works and one that is easily used is at the core of usability. A "hacker"[^1] might happily delve into the inner workings of a program if that is what it takes to get the work done, or even for the sheer joy of learning. They might practically swear by a program or system if they find it especially useful.

But anyone who lacks such enthusiasm, or simply the necessary time to spend on learning to use a product, will gravitate towards ones that are easier to use — sometimes even if they have to sacrifice functionality. For example, there are operating systems like Linux that are commonly thought to have many desirable properties, such as high stability or open source software support, and they are popular for critical or technical purposes such as hosting servers where functionality is key and the users are paid professionals. Yet only a small portion of home desktop computer users use them[^StatCounter2015]. Most PCs come with Microsoft or Apple operating systems pre-installed,

[^1]: In the sense of a person who is enthusiastic about something and what makes it work, and how to make it do what they want. Compare "life hacking": [http://en.wikipedia.org/wiki/Life_hacking](http://en.wikipedia.org/wiki/Life_hacking)
and changing something as fundamental as the OS is intimidating to the inexperienced, as it controls almost everything.

As such, methods for how to present complicated issues in easily understandable ways should help in making attractive designs. But what exactly makes one product more usable than another is hard to quantify, and how to improve usability is usually not obvious. Even mere familiarity is an important factor, nevermind things inherent to the design.

Therefore, developers hoping to reach a wide audience and not just "hackers" need to have usability in mind, and for that they need methods that both reliably improve usability in their designs right from the start, as well as reliably verify that a program or feature is sufficiently useable once completed. The field of Human-Computer Interaction satisfies these needs, and so was used in this project to improve the end result.

1.2 Objective

The stated goal is to gather a set of satisfactory requirements for an interface that teachers use to submit time table preferences, and then to implement these requirements to show that they are feasible.

These basic requirements follow from the nature of the project:

- Teachers should be able to use the GUI without (much) prior instruction.
- The GUI should allow the staff to save time compared to the existing methods of submitting preferences.
- Preferences made using the GUI should be in a standardised form that facilitates time tableing at TekNat.
- It should be possible to translate the standardised preferences into input to the existing program, or an upgrade thereof.

However, more detail is needed, especially before implementation can begin, which is why we turn to the field of Human-Computer Interaction.

1.3 Limitations

For a variety of reasons some aspects of HCI were considered safe to be ignored, or impossible to adhere to, with certain consequences.

- Normally the existing system is studied in the field as it is being used, in order to spot problems and opportunities. As real scheduling is a very rare activity on the teacher’s side, the closest to this that could be managed was asking interviewees to show or explain what typical usage was like. This still provides us with useful data, but is prone to focusing on what the interviewee feels are the main issues of the system while missing ones they personally had no trouble with or considered less important.
• Similarly, as Java Swing — the coding library of choice during implementation, see section 6.1.1 — has in-built functionality for usability-assisting tools, like screen-readers for the visually impaired, such functionality was also assumed.

• Java Swing also automatically ensures a visual design consistent with users’ systems, so aesthetic issues like what colors are appropriate or how to shape buttons such that they look invitingly clickable are largely assumed to be handled by the same in-built functionality.

• As the users are Uppsala University staff, it was further assumed that using English is not an issue, and also that there are no cultural issues, with this last being aided by the interface’s sparse text and imagery.

Aside from HCI, it’s worth mentioning that the initial intent of the project was purely to create a better interface for Hellkvist and Sjöstedt’s logic core, with user-friendliness being important but not the most interesting part. However, after discovering that a number of programs already exist that can perform automated scheduling — see section 2 — the focus shifted to an interface properly designed and documented with Human-Computer Interaction techniques. Because of this shift midway through, some of the initial work failed to follow HCI as closely as it could have.

Finally, due to the choice of implementing the interface as a Java applet, portability was assumed, resulting in little attention given to platform compatibility. See section 6.1.1 for discussion on this choice and its consequences.

1.4 Section contents

Section 2 covers related work, and programs with similar functionality to the intended interface. Section 3 explains design according to HCI. Section 4 goes over methods that are used in the project and ones that were considered but rejected. Section 5 briefly details the project’s results. Section 6 evaluates both the results and the methods. Section 7 contains some concluding discussion on the results and recommendations for the future.
Chapter 2

Previous work

2.1 Overview

Hellkvist and Sjöstedt [Hellkvist, Sjöstedt 2012] had an interface for their logic core, but it was straight to the database they used for testing their program, with little concern for being usable by staff without needing a manual or lengthy introduction. Citing section 7.2 of their report:

We recommend tuning the user interface to be more intuitive and helpful. While it currently implements the functionality required to construct the courses to be scheduled, it is not as easy to use nor as graphical as desired. The web site is at the moment a sophisticated interface to a database and that view should be changed to better fit the tasks of a teacher or scheduler.

This is followed by suggestions on what specifically could be improved: drag-and-drop functionality, a 2D grid display of weekdays and time slots, editing an event without leaving the page.

But there has been plenty of other work done in the past to assist with scheduling, or automate it to some degree, and we can certainly learn some things from such attempts. Here are a few examples of commercial software that handles the desired task, as well as some reports from similar projects:

- MosesKonto is a scheduling system made at the German university innoCampus, formerly “Zentrum für Multimedia in Lehre und Forschung”, for their own particular needs.

- Wise Timetable is a commercial program. It allows scheduling for all levels of education.

- Lantiv Scheduling Studio ditto, plus hospitals, conferences, and more.

- Cobi was an experiment in scheduling events with the help of crowdsourcing, which in this case consisted of people adding details like what events could not happen at the same time (in different rooms), what events should happen immediately after another, and so on.
• *On the Design and Development of User Interfaces in Interactive Scheduling Systems* is an older paper outlining things to keep in mind when designing an interface so it's usable by the typical users and not just the programmers. Decidedly HCI-related, if not formally HCI.

• *Display Requirements for an Interactive Rail Scheduling Display* is actually a paper about how to condense information into a useful interface for train drivers, so they can make more informed judgements about their schedule. Of interest from a HCI perspective.

• *Scheduling Home Control Devices* is a paper about programming VCRs. More specifically, it goes through the process of how to design an interface for that functionality that is intuitive yet powerful. It mentions using a common HCI technique, and has an interesting outcome that warrants comment.

MosesKonto is probably the most closely related project, having been made internally at innoCampus, for innoCampus.

### 2.2 Similar applications and products

#### 2.2.1 innoCampus’ MosesKonto

Developed at innoCampus as their University Course Timetabling software [MosesKonto, 2003], the site has this to say about the motivation for product (quoted without alteration):

> In the summer semester 2003, the MosesKontos was first used at the TU Berlin. Due to an extensive reorganization of the service orientated courses of the Institute of Mathematics and the consequent enlargement of the courses to up to 2,300 participants, the former organizers were faced with new organisational challenges.

> A task that was not thoughtfully beforehand was the division of students into the tutorials. Initial tests with the conventional batch or a first-come, first-served method proved to be time consuming, impractical and provided only slightly satisfactory solutions.

The resulting functionality is then explained accordingly:

> At the beginning of each semester, the organizers of the participating courses insert all needed information, all possible events and tutorials rooms and off-times of their events, using a web interface. Afterwards, the students register online for a place in a tutorial at the participating events. In addition, they can specify time preferences for their tutorials dates. These data were collected on a specific date and evaluated so that all interested students get assigned to non-overlapping seats in the small exercise groups for their chosen events. Among all the different non-overlapping solutions, our algorithm now determines one, in which the time preferences of all students considered the best overall.
They also claim that: "Criteria, such as for example time or space utilization priorities are modeled in such that they are demonstrably met perfectly — that is objectively there is no better solution[emphasis added]."

If UU wanted the students to be involved in deciding when lessons were held, this system would apparently work well. The interface itself gets little discussion, so presumably the functionality was what they were really interested in, similarly to Hellkvist and Sjöstedt’s work on the logic core.

Fig[2.1] is the view lecturers get when deciding on the timetable, and in fig[2.2] the view students get when viewing their course schedule.
Figure 2.2: innoCamp courses interface
2.2.2 Wise Timetable

![Wise Timetable Screenshot]

Figure 2.3: From Wise Timetables interface help page

Wise Timetable is a commercial product for "manual and automatic course scheduling over multiple weeks", compatible with EU university Bologna Process-compliant systems. It can export timetables to the iCalendar format, and would probably be a fine choice for TekNat's scheduling purposes. There are of course no publically available details on how they went about making their interface, as it is a commercial product rather than an academic report. The visuals can serve as inspiration for this project, though.

Fig 2.3 is an image from the official help page showing some details on what the software can do, and Fig 2.4 shows the program in the middle of automatically generating a timetable.
Figure 2.4: Wise Timetable generating a timetable
2.2.3 Cobi

Cobi: A Community-Informed Conference Scheduling Tool

Cobi comprises (A) community-sourcing applications that collect preferences, constraints, and affinity data from community members, and (B) a visual scheduling interface that combines community-sourced data and constraint-solving to enable organizers to make informed improvements to the schedule. [Kim et al., 2014]

Constraint solving is not used in this project but is an important part of automated scheduling, as seen in Hellkvist and Sjösteds’s report [Hellkvist, Sjöstedt, 2012].

Cobi is primarily aimed at the organizers and scheduling staff so they can easily make schedules and sort out conflicts; see fig 2.5. This is different from what is being attempted with this project where the aim is to give teachers an easy and useful interface to submit their schedule preferences, but something like Cobi could be a natural follow-up to this project to ensure both sides of the problem are solved.

Unfortunately the paper turned out not to be related to Human–Computer Interaction despite being found while looking for interfaces made with HCI; instead, there were events to be scheduled as part of an HCI event called CHI. There’s no mention of any techniques used. They explain their features and what they are for, but not how they arrived at them.

1Constraint satisfaction problems (CSPs) are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. Objects might be "sessions" or "attendees", while constraints might be "no overlapping in time/place". 
2.2.4 Lantiv Scheduling Studio

Figure 2.6: Lantiv Scheduling Studio student groups

Lantiv is another example of a commercial scheduling program [Lantiv]. There is no mention of their design methodology, but it seems polished and professional, with a number of features that could be useful for scheduling at TekNat.

Fig 2.6 shows Lantiv’s ability to assign groups of students to an activity at once, automatically displaying that activity in each student’s schedule. TimeEdit has that sort of functionality, so studying Lantiv more closely could be of interest in some future project.

Fig 2.7 displays Lantiv’s ability to link two different activities in different ways. The interface produced in this project ended up having a slightly similar feature in the ability
to determine the order of two given events.

<table>
<thead>
<tr>
<th>Course</th>
<th>Group</th>
<th>Instructor</th>
<th>Room</th>
<th>Length</th>
<th>Quanti</th>
<th>Sched</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Set 1</strong></td>
<td><strong>Sub Group 1</strong></td>
<td>John</td>
<td>Science Lab 1</td>
<td>01:10</td>
<td></td>
<td>02:20</td>
</tr>
<tr>
<td>Biology</td>
<td><strong>Sub Group 2</strong></td>
<td>Robert</td>
<td>Science Lab 2</td>
<td>01:30</td>
<td></td>
<td>03:00</td>
</tr>
<tr>
<td>Physics</td>
<td><strong>Consecutively</strong></td>
<td>Mary</td>
<td>Physcs Lab</td>
<td>00:45</td>
<td></td>
<td>01:30</td>
</tr>
<tr>
<td><strong>Set 2</strong></td>
<td><strong>Group 1</strong></td>
<td>Patricia</td>
<td>Library</td>
<td>01:00</td>
<td></td>
<td>02:00</td>
</tr>
</tbody>
</table>

This is how the set activities appear when they are scheduled.

Figure 2.7: Lantiv Scheduling Studio can group courses and order them logically
2.3 Research and reports

2.3.1 On the Design and Development of User Interfaces in Interactive Scheduling Systems

![Diagram](image)

Figure 2.8: In a nutshell

The design and development of scheduling systems generally centers on the functional requirements for problem solving; however, these function-oriented systems normally do not provide suitable or friendly user interfaces.

[...]

In order to build up scheduling systems that can provide users full extent of scheduling functions, we need to take into account human factors during the design and the development process in all aspects.

The grammar and wording made this paper [Yen et al., 1998] a bit hard to understand at times, and it mentions terms like three-dimensional Gantt charts without explaining them, but it gets some good points through even without full understanding. It is probably correct about scheduling systems at the time (1998) being more focused on functionality, leaving programs of the time unnecessarily complex and difficult to use.

The paper also suggests to first decide the categories of information you want to present, and the types of users you want to present to, in order to best determine how to present the former to the latter. As well, it makes the interesting observation that apart from the schedulers and schedulees there are also strategic users, who would get use out of a higher level view of the schedules, which would certainly be useful for this project or a follow-up.

In order to make the interface work for such users, some way to load all scheduled courses at once after the scheduling staff have finished would be useful, which in turn necessitates the ability to view these in a useful way. This might mean filtering by teacher, or course, location, etc., and seeing several months at once rather than the more "zoomed-in" view teachers and schedulers get. But this is all left to future work — see section 7.1.

Fig 2.8 sums the process up, though the devil is in the details as usual.
2.3.2 Display Requirements for an Interactive Rail Scheduling Display

![Train scheduling display](image)

Figure 2.9: Train scheduling display

While very different sort of scheduling, this paper [Tappan et al. 2011] had some interesting features and explanations, so it deserves comment.

It describes "Hybrid Cognitive Task Analysis", a process that could have been useful early on in this project. The process is described in the paper as such: "deriving a complete set of computer interface information requirements necessary to meet mission goals directly from operational tasks."

There is no mention of why they picked that particular method, but the prototype they made using it was easy to appreciate, being easily understandable, yet full of information and with the most critical bits — terrain and acceptable train speed — taking up proportionally more space. See fig 2.9.
2.3.3 Scheduling Home Control Devices

This paper [Plaisant et al., 1990] mentioned a HCI method they used, and had an outcome that seemed particularly interesting, so even though it once again is a different kind of scheduling it also deserves discussion.

The idea was to make a planning system for several home applications such that they all shared a central interface. In particular, VCRs had proven troublesome. The research proposed a few different interfaces, one using a typical calendar and 12-hour clock, another with a 24-hour clock, and the last with a sliding bar you set “flags” on to signify starting and stopping points.

That last one was easily the most unfamiliar method for the interviewees when it comes to deciding what time something would start and stop on — yet it was the most successful! Potentially because of being so unfamiliar, as that meant the users had no preconceptions, and therefore would get less frustrated when it behaved unexpectedly. However, without also being so intuitive it could be understood after a few seconds of looking, this would not have been enough.

As for the HCI method used, they videotaped participants and their interaction with prototypes, making note of their reactions — negative and positive — which is a common
method during the design process in HCl. It gets only brief mention though, and other than that the project has nothing to say about HCI or user-friendliness. Fig. 2.10 shows the final version of their design.
2.4 Conclusion

A number of programs and systems exist that could be adapted to TekNat’s situation, but none of them focus on usage of HCI in their development, so this project is distinct in that regard. Unfortunately, this also means there is little in the way of earlier experiences to learn from.

Some similarities can be noted in the features of the examples here — not so much the train schedule display, which performs a very different function, but the rest all have some form of calendar-like visualization. This makes sense; it is something familiar in the context of scheduling, and familiarity is an important consideration when designing.

There is also a slight tendency of having a "main window" placed towards the lower centre/right, with information and other bars along the top and left sides. Presumably this is because of English-speaking users and designers reading left-to-right, top-to-bottom, so their eyes first fall on specific areas of a screen, leading designers to place more important information in those areas. A quick google image search for "arabic interfaces" seems to support this idea — many are mirrored, so to speak — though a large portion are still left-to-right aligned.

On the other hand, the examples differ on whether a week should be displayed with weekdays as columns, with time of day or other content in the rows below, or the other way around. Those that used days as rows usually had a great deal of content in each day, and as such effectively used the greater horizontal space afforded by the majority of screen resolutions. Wise Timetable stood out here — the standard view is days as columns, time of day as rows, but there is another mode with the teachers as rows and the time of day as columns, with the day of the week instead being a tab selected at the top. This mode would probably be useful for scheduling staff, especially if you could easily swap the teachers as rows for rooms or courses as rows.

Ultimately, the polish and sheer amount of features existing applications have would be very difficult to try and match, as they have existed for years and have been worked on by more people. But formal design methods should make it possible to at least provide a solid fundamental design for the intended functionality, more suited for expanding upon in the future as the staff sees fit than any outside work could ever be.
Chapter 3

Theory

In this chapter, some underlying concepts of design are discussed. Designing could be thought of as a process going through four stages:

1. Requirements gathering — Determine what the system should be able to do.

2. User analysis — Determine restrictions and requirements of your intended users.

3. Prototyping — Implement functionality from the previous two steps in prototypes and test them.

4. Visual design — Ensure the visible aspects of the design aid in using the system.

The specific order is not usually strictly adhered to, as flaws revealed during prototype testing may be a result of poor requirements — perhaps worded wrong or not expressing the client’s actual intent — or perhaps a requirement turned out to be infeasible with current technology, or a prototype needs its functionality reworked before an acceptable visual design is possible. But these four phases provide a good framework for structuring design documents, and the field of HCI includes a variety of techniques that help ensure these steps produce something useful and usable, so they will be used to help structure the report.

3.1 Human–Computer Interaction

Human–Computer Interaction could, very roughly, be described as a field dedicated to figuring out how to design computer systems that are as effective as possible — allowing users to complete their task as quickly as possible, while remaining both powerful and user-friendly. The methods of the field can even be useful outside the context of computer systems, as much of the theory underlying HCI methods comes from other fields, especially psychology [A. Sasha Giacoppo, 2001].

It is fairly young as a field, being established in 1980 [Benyon et al., 2005, Preface], but has expanded rapidly as computer systems become increasingly ubiquitous, covering everything from smartphone applications to virtual reality training software. Producing a scheduling interface should fit well within the boundaries of its knowledge base. But before production can start, a design is needed, and to make a design, requirements are needed.
3.2 Requirements gathering

The initial requirements typically arise from what the product should do and who will use it. More detailed requirements can be gathered by then asking the intended users what they would expect from the product. This however requires the gatherer to know how to ask questions that produce useful requirements, and thus requires them to know what constitutes a good design. Benyon et al suggest three particular views should be taken into account in any good design — accessibility, usability, and acceptability:

Accessibility concerns removing the barriers that would otherwise exclude some people from using the system at all. Usability refers to the quality of the interaction in terms of parameters such as time taken to perform tasks, number of errors made and the time to become a competent user. Clearly a system must be accessible before it is usable. A system may be assessed as highly usable according to some usability evaluation criteria, but may still fail to be adopted or to satisfy people. Acceptability refers to fitness for purpose in the context of use. It also covers personal preferences that contribute to users ‘taking to’ an artefact, or not. [Benyon et al. 2005, p80]

To expand on and explain these:

- **Accessibility** - a variety of people will use the product, or desire to do so, but may be helped or hindered by various factors. For this project, it could be important whether the interface needs to work with screen readers, or use high contrast colors suitable for colorblindness. Other products might be concerned with how to allow wheelchair access, what price level is suitable, or the time of day a service is available; all things that might preclude access.

- **Usability** - Does the design allow users to do what they want? Is it an improvement over previous solutions? For this project, it is important that the interface is faster than handing in requests manually, and powerful enough for the scenarios users can be expected to encounter, while still being easy enough to learn.

- **Acceptability** - The design needs to be appropriate for its purpose and context. For this project: since the interface is meant to be used in a public environment, it should look professional, and not use overly informal language. A product aimed at children might want to use bright colours, and any symbols used need to be suitable for the intended audience; a floppy disk for saving would be unacceptable due to it being a several outdated reference — nobody uses floppy disks when saving files these days — but is acceptable because it is so common a symbol that is still understood.

Compare to the ISO 9241 standard for usability:

The effectiveness, efficiency and satisfaction with which specified users achieve specified goals in particular environments.

- **Effectiveness**: the accuracy and completeness with which specified users can achieve specified goals in particular environments
• Efficiency: the resources expended in relation to the accuracy and completeness of goals achieved
• Satisfaction: the comfort and acceptability of the work system to its users and other people affected by its use

It is clear from these that what the users want and need determines much of the requirements, so in order to know what must be taken into account for a given userbase, some amount of studying the users and what they do is necessary.

3.3 User Analysis

User analysis is about analysing users and adapting the design to them — not just the intended userbase for a specific product, but users in general as well. This means the analyst needs to understand the strengths and limitations of their particular users as well as people in general, so having an understanding of psychological concepts that apply to everyone will aid in the analysis.

For example, "Chunking", a concept describing how people split mental tasks up into more manageable chunks [Benyon et al., 2005, p339]. A phone number can be looked at as groups of two or three numbers, meaning 151230 is read as something like fifteen-twelve-thirty instead of one-five-one-two-three-zero. Similarly, a lengthy chore can be divided into smaller sequences, such as mopping the floor being split into:

1. Take out the mop and bucket from the closet.
2. Fill the bucket.
3. Wet the mop.
4. Mop the kitchen.
5. Mop the hall... and so on.

For this project, the interface can be assumed to need users to input information such as course codes, names, dates, and so on. To aid with chunking and make the interface feel more intuitively structured, the input fields for these bits of information should be separated by type, so dates and times are input near each other visually on the screen, or temporally in time, while names and course codes would not be similar enough to warrant such grouping.

When looking to design for specific users, the analyst may turn to interviews, which can be quite vague or very specific in their lines of inquiry, see section 4.2.1 or they may observe existing users as they work should there be an existing product that needs to be improved upon. Benyon et al recommend making use of a PACT analysis [Benyon et al., 2005, ch2] — PACT meaning People, Activities, Context and Technologies, so an analysis of those four things to produce requirements, see section 4.2.2 — or the more extensive Contextual Design method [Benyon et al., 2005, ch12-13], but in either case the end goal is a set of requirements tailor made for a fairly specific set of users, such as the staff at TekNat. It is important to learn the terminology of the users, both to avoid
misunderstandings on the part of the analyst and to ensure the product uses relevant words and phrases, and even imagery.

Whether by interview or observation, it is important to take notes, or record everything with a camera or microphone. Better yet, combine them to cover weakness of one with strengths of another; such as it being hard to make out what is being said in a recording, or notes missing some important statement in the rush to write. Interviews tend to get especially informal and stray from the subject, and as people generally want to get along it can be hard to confront or even notice biases in preferences or emphasis. When reviewing a conversation from notes or footage it is usually easier to handle such issues.

There are many more techniques for gathering the detailed requirements intended with this phase, such as brainstorming, artifact collection, probes, scenarios... but the end result should be enough requirements to produce a prototype suitable for testing.

3.4 Prototyping

The prototyping phase is not just implementing the requirements so far in code and then testing the result, but also for specifying further requirements and discovering unforeseen problems in the design before too much time is spent on the real implementation. Usually this involves "envisionment" [Denyon et al., 2005, ch8], meaning that the ideas developers and users have are made tangible to some limited extent. For this purpose the developers should first turn to paper prototypes — sketching what the product may look like once complete, see section 4.4.1 — or other lo-fi solutions, in order to try out prototypes that are quick, easy, and cheap to make and modify.

Techniques in this phase, such as the aforementioned paper prototypes, will all try to show users what a typical usage scenario might be like — as the developer has envisioned it. This helps to clear out misunderstandings on both sides, as something expressed in an interview during the previous phases might have meant very different things to the interviewer and the interviewee. For instance a developer adding a transparent scrolling text element after an interviewee expressed wanting "clear marquees", when they really wanted a scrollbar that stood out visually.

At some point full implementation begins however, and testing the hi-fi prototypes is just as important as the earlier ones, though the role of the developer may shift to one of more passively observing users as they attempt to use the product and seeing what issues become apparent, as opposed to actively discussing the features. In the final, real version, the developer will not be available to sit with each user and explain the product, so it must be able to be usable without the developer directly helping before release. During this stage the broad underlying design ideas should be in place, so improvements will tend to be more in the area of visual design.

3.5 Visual design

Perhaps the part most people associate with the word "design", this phase of the process is not so much pure aesthetics — what looks good — as it is about visual issues, such as how to lead the attention of users so they notice what they need to notice, providing the right amount and type of detail for the situation, how to provide feedback when a
task is complete, etc. While making an attractive interface is good, as it makes usage more pleasant and draws in new users, the trick is in combining what makes an interface attractive with what makes it better in terms of usability.

As such, developers that want to follow good design standards will probably not need to know what colours match the season, but should know about common ways to ease usage. For example the "midaught button", which refers to buttons that are near the edge of the screen — such as the minimize/maximize/close buttons on the upper right of many application windows — which should preferably touch the edge of the screen, such that even if a user makes a swipe with the mouse that would have sent their cursor a mile off-screen if it weren’t restricted, they will be able to click one of the buttons without moving it back in a few pixels. It is a small detail that only helps to occasionally reduce strain from precision work with the wrist, similarly to just making buttons as large as possible, but many small details like that make for a pleasant to use interface.

First the design needs to allow users to understand how to actually use it though. Metaphors [Benyon et al., 2005, ch9.3] can be very useful here. Consider the nearly universal symbol for save buttons: a floppy disk. While these days its purely out of familiarity, since even people who have never seen a floppy disk have probably seen plenty of save buttons and thus know what it signifies, at one point saving data was something you might do by moving it to floppy disks that you could easily store and organize, which made the symbol a metaphor for that. In fact even the phrase "save your data" is a metaphor; before computers, "save" probably just meant saving as in rescuing, or saving as in money in the bank — both meanings can be very apt for information you want to keep around, but by now it has taken on a meaning of its own.

Such things are invaluable for aiding in making a design intuitive. For the interface in this project, since it has to do with scheduling, it felt intuitive to use a style similar to calendars to make it more immediately obvious what the user is looking at when first starting it. A widely used term for such design where it is intuitive what something is or how it is used is "affordance". One memorable example often given for this is doors: handles are for grabbing, so a door with handles "affords" grabbing and pulling, while a door with no handles (or knobs or anything) "affords" pushing [Benyon et al., 2005, ch24.4].

Whether it is done by metaphor or just making "wrong" actions impossible, like the door with no handles, a good visual design guides the user without need for verbal or text-based explanations. Much of the testing during the prototype stage can reveal aspects of the product in need of such improvement, though figuring out what exactly to do can be difficult without existing examples to look at.

3.6 Conclusion

Going over the entire theory of HCI would require this report to be an entire book, so while it would be relevant and interesting to cover things like how memory and attention works in people — or what avatars are good for, or what makes mobile apps need different design approaches, or the importance of keeping in mind social aspects like culture or group beliefs and behaviours [Benyon et al., 2005] — the above will have to be considered some kind of brief summary on the aspects most relevant to this project. Understanding
the theory is important, but to make good use of it a developer needs more specific ways of making use of that understanding, in the form of techniques and methods.
Chapter 4

Method

In this chapter, methods and techniques that are used in designing are explained, using the same four phases for structuring as in the theory section\[3.1\], requirements gathering, user analysis, prototyping, and visual design. Contextual Design gets separate mention however, as it is a complete set of methods for making a design.

4.1 Contextual Design

A full Contextual Design, or CD, will take the designer(s) through the four phases — requirements gathering, user analysis, prototyping, and visual design — all on its own. It entails gathering a lot of detailed information about the context and everything generally surrounding some piece of work, typically having the team visit the workplace and observe the users as they work, in order to identify what can be improved as well as what needs to be preserved. It was deemed as too ambitious for this project, as it makes use of a wide variety of skills and thus works best with a whole team, but some aspects of it were still used.

When should Contextual Design be used?

Contextual Design is written as a method for the design of generic products across a particular customer sector. For example, the aim might be to develop an administration system for small to medium-sized hotels such as the one we focus on in these two chapters. As well as keeping data customer-centred, CD supports designers in collecting data from different organizations and sites and distilling common elements. CD is also intended to be used by a design team, which means that its models (for example) are helpful for sharing data and interpretations as well as recording them. This does not mean that CD cannot be used for smaller projects, but you are likely to find that using all the techniques recommended on a one-person project for a small company is not worthwhile. Indeed, the method is designed to be modular. [Benyon et al. 2005 p273]

The first stage of Contextual Design is the Contextual Inquiry, where you "go where the customer works, observe the customer as he or she works, and talk to the customer about the work. Do that, and you can’t help but gain a better understanding of your
customer." [Benvon et al., 2005, p274] A proper Contextual Inquiry would stick around
and observe the staff at work even when their tasks were unrelated to scheduling, in
order to get a better understanding of the context. For the project this was considered
too time-consuming for one person to do. Moreover, it was not feasible to watch the staff
perform the task at hand, since it is a relatively rare and quickly completed task. Instead,
asking to see what writing a schedule request might be like during interviews was used as a compromise.

The next stage of CD is modelling the data gained from the inquiry in various ways
— flow models, sequence models, artefact models, cultural models, physical models —
and noting areas that can be improved. None of these were used directly in this project,
but the idea of noting intents and issues of specific parts came in handy when discussing
paper prototypes with interviewees.

After that comes the consolidation stage, where the team goes through their models
together and compare the results, in particular any problems and solutions spotted during
the earlier stages, then just try to figure out design elements that take all of that into
account. With a one-person team there is nothing to consolidate with, so for the project
the closest parallel to this stage would be when interviews were reviewed and common
opinions identified, and potential solutions noted.

Similarly, the shared vision stage is not applicable without a team; with nobody else
to share it with, figuring out what the vision actually is becomes a bigger issue than
communicating it. Considering the data acquired so far, what sort of image should the
interface have? Given the academic setting, probably something more professional and
business-like, rather than warm and inviting. What is the main functionality? The ability
to specify schedule preferences (while still giving the scheduling staff as much to work with
as possible), to save them for later, to display existing preferences in an understandable
way, and so on.

CD goes on through a few more stages, like constructing storyboards and a User
Environment Design, but as these were not used in the project at all they will be omitted.
They give the team an understanding of what the design should be able to do and the
image it ought to have, enough that the design can finally start taking on more definite
form. Sketches and descriptions — like paper prototypes, see section 4.1.1 — can be
made and brought to the customers in order to get their input, and then all that remains
is implementation, testing, and rollout.

But since this project did not use a full-on CD, it had to use other ways to get a
coherent design, and went through the four phases with a mix of methods, including
those from requirements gathering.

4.2 Requirements gathering methods

There are many ways of getting the initial requirements in Human–Computer Interaction,
but interviews are probably the most common and effective way.
4.2.1 Interviews

In a sense, even a testing session is a kind of interview, so the methods described here may be used throughout an entire project — but the most extensive interviews are at the beginning. They can be conducted in different ways, and HCI separates them into three types: structured, semi-structured, and unstructured [Benyon et al., 2005, p152]. Structured interviews make use of many specific questions, while unstructured interviews have few and very general questions — they may even skip questions altogether and just talk about the subject — with semi-structured being somewhere in-between, perhaps asking about specific features but allowing the talk to stray from the subject more, if it seems at all relevant.

The advantage of more structured interviews is that they provide the desired data; the disadvantage is that anything the interviewer did not realize they desired to know, and thus failed to ask about, never comes up. In less structured interviews, the problem lies mainly in keeping the conversation going in the right direction, such that enough useful data is assembled. The type that ought to be used may vary between interviewers, interviewees, or perhaps even the current mood of either, all according to what fits best. For example, in this project semi-structured interviews were used, as deciding on what questions to ask in a fully-structured interview would require more experience in the field of design than was available, while unstructured interviewing would require more practice at keeping a conversation focused. According to Benyon et al, semi-structured interviews are the most commonly used form [Benyon et al., 2005, p152], so they should be a good starting point for a student of HCI.

Regardless of the form used, HCI recommends taking notes of what is said, or recording audio or even video, but the more the better. It helps to have a second source in case something heard or written is unclear, and video or audio can be especially useful in providing a different perspective for interviewers, who normally cannot see or hear themselves in the same way interviewees do, but the reason for recordings or notes in the first place is mainly reviewing later to extract specific requirements. Having another person along to take notes or ask questions can be very helpful.

Gathering data via questionnaires is another option, and they work similarly to a fully structured interview with how they ask very specific questions, but while questionnaires can reach a great many people they take a great deal of time to prepare if they are to produce useful data, and the questions need to be very carefully designed to avoid ambiguity or misunderstandings in the responses [Benyon et al., 2005, p156]. More experience with the subject may have made them a more feasible option, though even then it is likely that the low response rate questionnaires get combined with the relatively few people (for questionnaires that is; less than 1000) working at TekNat would have produced too low a sample rate for usable data.

A method that did see use is the PACT analysis.

4.2.2 PACT; People, Activities, Context, Technologies

PACT is a framework for approaching design by considering four perspectives [Benyon et al., 2005, ch2]:

1. What kind of People will be involved — their skills, dexterity, economical situation,
anything that is relevant.

2. What their Activities will be — what they currently do, how they do it, what any new designs will require of them...

3. What Context will those activities be conducted in — in private, in a foreign culture, formally, downtown, in winter, around children...

4. What Technologies that will be used — platform, software, physical tools...

For this project, these perspectives are something like this:

1. People: University professors, scheduling staff, teaching assistants

2. Activities: Making new schedules, modifying old ones, reusing old ones, walking into the office of the scheduling staff to discuss specifics...

3. Context: University campus, in private sending e-mail to the staff, or in person talking to them directly.

4. Technologies: Access to computers and the internet is a given. There is a wide variety of platforms available, including mobile devices. E-mail to the staff, often used alongside the in-person visits.

From this analysis alone, many requirements and possibilities already become clear. For example, the interface should be platform independent, but on the other hand it can rely on the user having internet access, while it also needs the ability to save a schedule and e-mail it or re-use later. A PACT analysis makes a good starting point, but this project used interviews and paper prototypes (see section 4.4.1) to get more specific requirements. An alternate approach would be to split each perspective further, such as considering People in terms of physical differences, educational differences, personality, or whether their Activities are complex, safety-critical, need coworkers, and so on. Many of these are implicit or unconsciously taken into account, but a more formal PACT analysis would try to detail them all.

It is also possible to get more specific with the method by considering more specific People, real or imagined, and exactly what Activities they would be involved in, and of course any more specific Context or Technologies surrounding them. This would make the analysis somewhat similar to a different method: User Personas.

4.2.3 User Personas

The User Personas method was actually rejected for this project after due consideration, but is commonly used and as such warrants discussion. It is named after the Personas the designer(s) construct, which could be said to be imaginary users, and are meant to be kept in mind when deciding on design features in the same way as real users. For example, consider the Persona of Lazlo, an imaginary nearsighted physics professor. They would want text fields and buttons fairly large and well spaced out, and would not be very interested in the ability to set up a script to modify saved preferences. In this way
the design team can come up with specific features that would not have been thought of otherwise.

But as "The persona’s new clothes" [Chapman & Milham, 2006] argues, there are some issues with the method. Personas, especially entirely imaginary ones, are not falsifiable — a writer of a persona can simply dismiss contradictory data, such as a real user not wanting a specific feature, as not being relevant to their persona, and this dismissal cannot be said to be incorrect. Moreover, they are unlikely to match any real person due to the level of detail needed to create specific requirements, raising the question of who exactly is being designed for.

"Personas Considered Harmful" [Nick De Voil, 2010] expresses similar concerns:

The persona is not intended to be an average user, or to capture in any statistically valid way a set of data points. It is a design artefact intended primarily as a communication tool; by making explicit and documenting a specific set of goals and behaviours, the persona gives members of the design team a common focus.

[...]

Made-up people are carefully composed and by their very nature can only have problems that the designer already knows about.

These issues are best improved by doing extensive research into the intended user base [Usability.gov], but in this one-man project the intended users were known and could be relatively easily reached, so it was feasible to ask them directly about design decisions. With this, and the lack of a team that could make use of the personas to help communicate ideas and issues, dropping the use of personas was deemed the most sensible choice.

4.2.4 Other methods

There are many more methods used in requirements gathering, generally studying the existing application that is already being used in some form, with current usage patterns being the main source of data on what needs improvement and what already works. However, the current scheduling process is completely manual and infrequently used, so such methods cannot be used for this project.

Fortunately these methods are typically some variation on asking people what the current application is like — “Tell me about your typical workday”; “Tell me three good/bad things about the system” — or observing them as they use it, so the interviews can make some use of the underlying ideas. Later interviews and prototype tests made use of aspects such as watching people as they work in order to spot problem areas, or missing features neither developer nor user previously thought to bring up; see the section on interviews, section 4.2.1, for more on that.

Regardless of method used, eventually the designer(s) will need to go from gathering requirements about the desired product and its functionality, to gaining an understanding of the intended users of the product. For that they will need methods for user analysis.
4.3 User Analysis methods

While methods like PACT and personas have already provided some awareness of the intended users’ desires and what they are like, understanding peoples needs and wants on a deeper level will help ensure they get what they need and not just what they ask for.

4.3.1 Access, Usability, Acceptability and Engagement

These four factors were explained in section 3.2 and provide information on their own, but for a more detailed evaluation they can be broken up into more specific aspects [Benyon et al. 2005, ch4]:

Detailed factors

- Physical exclusion — Access may be physically hindered; if the intended users are unable to use the design because it requires them to be a certain age, or to be able to get through a specific locked door, then the design has failed to properly take this factor into account.

  The scheduling interface has some issues like this, as it requires a specific version of Java, preventing any user without that version from using it. Had this issue been known beforehand it would have been avoided by using some implementation other than a Java applet. On the other hand, one tester commented on the bright colours being easy to distinguish even with colorblindness.

- Conceptual exclusion — Access may be hindered by the design being hard to grasp conceptually. If users need to know programming to understand the instructions, commands, and so on, then any users that do not simply will not use the product.

  For the scheduling interface this was occasionally an issue, particularly the week preview which did not look like anything familiar nor react or otherwise give much feedback on user actions — unless they had already created one or more requests, but interestingly most users tried to interact with the week preview first of everything they did in the interface — but these problems were revealed after prototype testing and could then be solved.

- Economical exclusion — Is the design affordable? Is its prerequisites?

  Fortunately not an issue for this project as it and Java are available for free.

- Cultural exclusion — Does it use jargon appropriate to the users? Are any metaphors (visually and otherwise) understandable to them?

  Difficult to evaluate without being part of the staff and knowing all the jargon intuitively. One recurring issue is what exactly it meant when users made a "request" in the interface; was it something they were guaranteed to get, how specific should it be, and so on.
• Social exclusion — Is it socially acceptable? Can it be used in any setting, and at appropriate times of the day?

In appearance the designed interface is inoffensive, but some testers felt that the current process of meeting the scheduling staff in person and discussing their schedule worked fine, and worried that the outcome would be worse when using an impersonal interface instead. Hopefully this will not be an issue, as it will still be necessary to contact the scheduling staff when submitting a requested schedule, at least until the entire scheduling process is automated.

• Visibility — Can people see what functions are available and what is happening? Sound and touch are also important here.

This came up in testing as well; it was difficult to interpret what the week list preview was used in the early prototypes, so it was improved with highlighted outlines as users clicked on or moused over weeks, making it clear something was happening with them. Sounds could have helped even further, such as the noise of changing a page when moving between weeks.

• Consistency — Keep things consistent. Unless you’re deliberately using inconsistency to draw attention.

Easily adhered to with Java Swing, the coding framework used, providing systemspecific visuals; see section 6.1.1 for more on it.

• Familiarity — Use familiar (to audience) symbols, systems and language. Or if this is not feasible, a suitable metaphor like the old floppy disk as a save symbol.

This was the main reasoning behind the calendar-like visuals.

• Adaptivity — The learning curve for different people, outside factors that may affect usage such as sun glare on a screen used outdoors...

The interface does allow for system-specific visuals as mentioned above, and will work reasonably well when resized, but is otherwise not changeable.

• Identity — A sense of authenticity. Users should not need to worry about the legitimacy of your design.

Here, too, the system-specific visuals should help, as well as the Uppsala University logo space in the upper left corner, where it is usually displayed for official webpages hosted by UU.

• Affordance — It should be clear what things let you do. A handle can be pulled, a large flat surface can be pushed, a raised border looks clickable, a splash of colour draws attention.

One of the main ways the aforementioned week preview issue was improved was in this way.

• Navigation — Provide support to enable people to move around the system easily — the structure should probably be straightforward and hierarchical so a mental model of the design can be easily assembled by users, different experience levels should be
catered to with easy direct access to common functions while still allowing them to
be reached in a logical step-by-step approach, etcetera.

The buttons and screens are fairly straightforward and easy to grasp conceptually,
and the inclusion of keyboard shortcuts should make usage quicker for experienced
users.

• Control — It needs to be clear who or what is in control, and of what exactly.
  If one action causes something seemingly unrelated to happen, users will no longer
  feel confident their other actions have the effects they expect. Suppose the interface
  automatically sent an email to the scheduling staff whenever a user saved their
  schedule requests — the intent might be to provide early drafts and aid in quick
  iteration, but it would make users feel like the program went behind their back,
  and they will most likely not trust it again.

• Feedback — The interface should give appropriate feedback.
  When the design must do something unintuitive, perhaps due to errors or safety
  standards or other things outside the design’s control, it needs to be clear that
  something was done, what that something was, and why. An example would be
  the week preview, which became considerably more intuitive with the highlighted
  borders on mouse over and clicking.

• Recovery — Enable quick recovery from actions and errors, like cancel buttons or
  even just hitting the escape key undoing the last event.
  An undo function would be useful, but is missing in the final prototype; it never
  came up in testing.

• Constraints — Ensure users cannot do undesirable things too easily, such as dan-
  gerous operations needing additional confirmation.
  There is no prompt when hitting the delete button, but the size of the buttons
  should minimize the amount of misclicks.

• Flexibility — Multiple ways of doing things. Allow minimal menu-based input to
  complete tasks, while also having a graphical interface. Personal user customization
  is a bonus.
  Mainly supported with the inclusion of keyboard shortcuts.

• Style — Attractiveness. Keep the intended audience in mind.
  Considered low priority as the prototypes focused on functionality and usability,
  but would be more important for any version intended to be fully used by the staff.

• Convivial — Polite, friendly, generally pleasant communication from system to user.
  Barely applicable; the interface never needs to address the user directly. Labels for
  buttons and input fields are straightforward and suitably phrased.
It proved useful to go through these more detailed factors during the early stages of the project to be reminded of potential points of failure, rather than only evaluating them after completing a prototype. They cannot cover every base — for example, the week preview was not obviously going to be a problem when it was first implemented, and only once testing began were the issues revealed — but they do help with ensuring the design keeps different user needs in mind early on, which saves time during the testing and implementation stages.

Mostly though, user analysis is done as part of interviews and testing, where you can speak to or observe users directly to see what needs and preferences they have. Once the initial interviews were done, paper prototypes could be made and discussed, followed by implementing functional prototypes.

4.4 Prototyping

The prototype testing stage can begin when a design has enough content that it can be shown to testers and have them interact with it. It can be done with a high fidelity (hi-fi) prototype, such as a more or less fully implemented and functional prototype, or one that is low fidelity (lo-fi), such as a set of sketches that the test conductor moves around for the tester as they use the prototype.

HCI emphasizes the importance of prototyping early and often, as well as testing with real users and not just dedicated testing staff. Prototype testing is more about seeing what features in the design needs work and what seems fine than finding bugs, though of course those are still noted and fixed.

The most common method for conducting tests is the "think-aloud protocol" [Benyon et al., 2005], perhaps because it is quite robust and difficult to get wrong results out of even for beginners [J. Nielsen, 2012]. It consists of observing testers as they try out the prototype, usually directed by a set of tasks created by the test conductor in order to get feedback on specific areas or cover as much of the product’s functionality as possible, and as they use it they think aloud, saying what they are trying to do and what their thoughts are on the design, how it hinders or helps.

Throughout this process the test conductor takes notes of what happens — are they trying something the designers failed to think of, is some part of the design going unnoticed, and so on — as well as the verbal feedback. Assistance is kept at a minimum, since needing to explain the design is usually a sign that it should be changed to be clearer. In practice though, the conductor will almost certainly have to prompt the tester a bit in order to keep the testing going, so when this inevitably happens it is important to give directions or ask questions in a proper way [J. Nielsen, 2012], which is where any conductor new to the testing process is most likely to introduce biases and other errors.

It is important to realize that the proper way to ask questions is not just about how to get useful answers, but also how the tester and conductor interact. Politeness goes without saying, but professionalism can actually be counterproductive, as a good method for getting a tester to expand on a question they have asked is to “um” and “uh” [Kara Pernice, 2014]: simply asking them to expand or what they mean could make them reluctant to ask again, if they feel put-upon. There are also ethical concerns:

Serving as a test user can be very distressing, and you have definite respons-
ibilities to protect the people you work with from distress. We have heard of

test users who left the test in tears, and of a person in a psychological study

of problem solving who was taken away in an ambulance under a sedation

because of not being able to solve what appeared to be simple logic puzzles.

There's no joke here. [Lewis & Riemer 1993, ch5.1]

In short, there is more to testing than just watching a user, but it is very useful. Early
testing is perhaps the most effective, as changing a sketched design is far easier than one
implemented in code, so lo-fi prototyping is given serious attention in HCI.

4.4.1 Paper prototyping

Lo-fi prototypes are usable throughout the design process as they are quick to make and
easy to change. Paper prototyping in particular is deceptively simple: draw ideas of
how a design should look, whether on physical paper or digitally, then show these to the
intended users and discuss the prototypes with them [Benyon et al., 2005, p.187]. The
project made use of paper prototypes early on and went through considerable changes
that might have been difficult to implement if they had been left for a later stage.

Despite its simplicity, paper prototyping is remarkably effective. As it is a drawing,
any necessary adding, removing and changing of things is quick and easy. Instead of
requiring extensive programming for every change, feedback can be incorporated even in
the middle of an interview, allowing for further feedback on the proposed changes.

Interestingly, testers will respond in different ways depending on how detailed the
prototype looks [Benyon et al., 2005, p.185-186] — a sketch on paper gets responses
focusing on functionality first, whereas more polished mockups often get reactions like
they are attempts at the final product, and get critique on details like colour choice. Such
feedback is important in later stages, but barely usable early on in the design process
where there are only mockups, instead of more or less functional prototypes. This effect
is strong enough that when making a paper prototype digitally, it helps to spend some
time finding a way to fake a pencil sketch-look. There are some existing commercial
programs with this ability.

Overall, the process is quite similar to other kinds of prototyping, so moving from
paper prototypes and other lo-fi alternatives to more or less fully implemented functional
prototypes is straightforward. As the testing goes on and functionality is completed, the
focus will naturally move from functionality to visual design, though major issues there
have likely come up and been addressed to some extent already.

4.5 Visual design

When it comes to spotting visual design issues, the best way is to observe users and what
they do, so the method there is largely that of prototype testing. More advanced ways of
observing the users exist, such as determining which parts of the screen the eyes focus on
— vital to website ad placement — but nothing more advanced than manual observation
and discussion was used in this project.

Knowing some useful concepts before the design was started helped avoid issues in the
first place, for instance the “mile-high button” concept mentioned in the theory section.
It could not be used directly, as the interface is confined to the user's web browser window and as such does not touch any edge of the screen, but making buttons larger and thus easier to click on when the space was available was an obvious and easy way to ease usage. Similarly, later versions of the interface had better affordance, such as the week blocks in the week view lighting up when hovering over them, making it much more clear they were clickable.

There are some areas where possible improvements are obvious however, left undone due to time constraints.

- There are no icons anywhere. Icons are not obligatory, but can be very useful as a visual aid or quick explanation for an interface element, like the familiar save icon.

- Some input variables could not be easily explained with just their name, and thus received mouse-over tooltips, which were only moderately successful at best; about half the testers missed, ignored, or were confused by them.

- There is some unnecessary information presented; times of day beyond those any staff would schedule, and more weeks than are likely to see use.

But in the end, a functional interface was produced, and the evaluation of it and the process used to reach it follows in the next two chapters.
Chapter 5

Results

5.1 Overview

Creating a useful interface requires the designer to first find out what the users need, which is typically done via interviews with people from the intended userbase. In this project various staff members at TekNat were asked to participate, as the teachers and scheduling staff there were expected to be the primary users, even if other departments or universities could potentially make use of the interface as well.

There were several sets of interviews and tests, done such that the results of one set shaped the next prototype, which was then evaluated in the following round. The participants differed in each set, with some staying between one and the next, but most being new. Due to this, prototype changes could be evaluated relative to how they compared to the previous version, as well as how they stood on their own.

The first set of interviews asked the participants what they might want in a graphical interface for the scheduling process in order to get a starting point for a design. After that, participants were asked to give feedback on proposed designs, usually in the form of paper prototypes or mockups. Finally, the last sets consisted of testing functional prototypes.

5.2 Requirements gathering

To determine the initial requirements, both teachers and scheduling staff at TekNat were asked about their experience with the current scheduling process to try and get an idea of what the finished interface ought to be like. Eleven individuals were interviewed. Notes were taken during each interview, and once the interviews were complete, these notes were sorted through, combined, and summarized, with the main points being:

- Users should have a variable amount of control, allowing both very specific requests and very general ones.

- However, they should be encouraged to be as general as possible, to make scheduling easier for the scheduling staff.

- The existing system often results in a lack of detail for the scheduling staff.
• It ought to be possible to reuse previous schedules.
• No manual should be necessary.
• Sometimes specific equipment is needed, so the ability to request that is desirable.

5.3 Prototypes

![Image of a paper prototype]

Figure 5.1: Paper prototype; sketch of proposed design

After the requirements were gathered, prototypes could be made, in particular paper prototypes such as fig. 5.1. Easily modified and focusing on functionality over appearance, sketches like this were shown to interviewees, evaluated, and changed according to feedback until a reasonably stable vision was achieved and implementation in code could begin.

This particular prototype displays some ideas that went unused in the final concept, which gets discussed in chapter 6.
5.4 Final result

![Final result](image)

**Figure 5.2**: Final prototype. Three requests are on display: a lecture and a lesson at any normal schedule time in Monday through Thursday, and a laboratory session on Friday. The visuals match the environment of Windows 7.

Fig [5.2](image) is the final prototype and considered a completed version of the interface, suitable for official use with minimal tweaking. Visual elements such as button shapes automatically change according to the operating system; here the visuals for Windows 10 are seen.

The design is meant to look a bit like a calendar: the boxes along the top show weeks with individual days, while the main part of the interface shows the contents of each weekday.

The colored boxes in each weekday represent a request created by a teacher. These only represent a potential timespace the actual session will take place in; it is left to the scheduling staff or automated software to decide when exactly within the designated time that a session takes place, and what the location should be.
Staff can save and load entire requested schedules, creating a file that can be reused later, passed around to and viewed by other staff, or edited directly.

Figure 5.3: Final prototype’s “New request” form. Here the visuals match the environment of Gnome/Linux.

After clicking the "new request" button, or double-clicking any existing request, the "request form" comes up. This form for making new and changing old requests is mostly straightforward:

- Session type and acceptable days are drop-down menus.
- Course code and the like are freely typed in.
- Color is purely visual, but automatically changed according to session type, making it easier to keep track of what sessions are in a week.
- Unique ID can be used to make requests that span more than one week, or are otherwise more complex than the default controls allow for setting up.
- Happens-Before allows setting a strict order two or more sessions happen in. For example, if the lesson in fig 5.2 has Happens-Before: 2, and the lecture has Unique ID: 2, the lesson must happen before the lecture.

- Comments are just free text, either for addressing other staff or making notes for oneself.
5.5 Conclusion

This project has decisively shown that automating the scheduling process can be done with a user interface suitable for the staff at TekNat. The specific goals, as seen in section 1.2, went as follows:

1. Create a GUI that is usable for making real schedule requests.

2. The GUI should provide its output in a standardized format, preferably such that the existing logic core or an upgrade thereof can take it as input.

3. It should also be usable without prior instructions.

Goals 1 and 2 can be considered successfully reached. Though the output isn’t immediately usable as input for the logic core, it would be relatively easy to change the output into XML, and updating the logic core accordingly if need be.

Goal 3 stumbles due to the technical problems had with Java. Users having to set permissions or alter policy files manually is unacceptable from a usability perspective, but there is no clear way to get rid of the issues without starting over in a different language, despite searching for one over several months.

With this, the interface is considered complete. The next chapter goes over the design process in more detail, as well as what future changes might be desirable.
Chapter 6

Evaluation

In this chapter the choices made throughout the project are explained and discussed, such as what programming language was chosen for the interface, or what changed in the design throughout the project, and why.

6.1 Language choice for making the GUI

It was necessary to decide on a programming language before coding and implementation could begin, of course. A number of languages were considered, primarily based on these factors:

1. The interface ought to work on multiple platforms; in particular, it needs to work on the computers staff use on and off campus. Mobile platforms would be a bonus.

2. The interface ought to work acceptably fast even on the campus terminals.

3. Making a working interface should be relatively quick, and the time spent on coding or learning how to work with the language kept reasonably low.

"A Comparison of Programming Languages for Graphical User Interface Programming" by Phillip Kevin Reed at University of Tennessee – Knoxville [P.K.Reed, 2002], helped inform these choices somewhat. Though a relatively simple paper, with the author’s predisposition towards one of the choices showing clearly, it gives some pointers on the three languages compared in it.

Languages considered:

- C/C++.

C/C++ was mainly rejected for being a low-level language; it has all the power one could desire for making an interface in, but implementation would take much longer. Security was also a concern, as users would be required to manually download, install, and run a program on their computer, but considering it would be issued officially by Uppsala University this would most likely not have been an issue in practice.
• Erlang.

Erlang is not known as a language used in making graphical interfaces, and was more of a personal favorite than an appropriate candidate. For a more computationally intensive task Erlang would have been a strong contender, given the ease of parallelization in it and personally considering it easy to work with, but ultimately the lack of experience in making anything using graphics in Erlang made it an impractical choice.

• Dynamic HTML.

Dynamic HTML requires the user’s browser to be up to date. Not a problem for most modern browsers which auto-update, but Internet Explorer users may opt not to [Microsoft, 2011]. As Internet Explorer updates are adopted slowly by users [W3Schools], anything relying on newer versions of it is likely to exclude a large portion of the population, making DHTML a dubious choice. On the other hand, the intended users of university staff might be quicker to install updates or more likely to use other browsers than the average user. In hindsight, it would have been wisest to ask the IT administration staff about the matter, but for general users, at the time of writing roughly 23% still use Internet Explorer [StatCounter], with the remainder being mainly Chrome and a small bit of FireFox.

• Flash.

According to the official Adobe statistics, Flash adoption rate is around 98% [Adobe, 2011], suggesting it would be effective even in a multi-platform environment like the university. At the time of choosing a language I was under the mistaken impression that the lab terminals did not support Flash on webpages, so in the interest of easing development, much of which would be on those very terminals, Flash was dropped. Additionally, Apple does not support Flash on their handheld iOS platform [Steve Jobs, 2010], so any future plans to make the GUI compatible with smartphones or other handheld devices would be hindered.

• Java Server Pages.

Using JSP would mean users get prompted to allow each operation when saving or loading, rather than having to give blanket permission like regular applets do, but it would require a server to generate pages for users. Setting up a server for the application at Uppsala University seemed like it would take too much time, and the inability to use the program without an internet connection — unlike an applet which can be manually downloaded — seemed an unnecessary limitation.

• Java applets.

Java applets need to be “signed” or have the user manually allow them to have access to their computer in order to do even simple things like save and load prompts, but they work entirely clientside, unlike Java Server Pages. They also do not require manual installation, much like Flash, as well as being nearly universally adopted, so accessibility is greatly eased.
There are also plenty of other languages outside of those in this list that could be used for a project like this, but the time to learn a new language from scratch seemed prohibitive. Ultimately, Java applets were picked as the most suitable approach, in large part due to previous personal experience with Java.

6.1.1 Results of choosing Java applets

Java proved a fair choice for coding the interface, with Java Swing\(^3\) providing functionality like GridBagLayouts, which allow a good amount of flexibility while being more high level for graphical manipulation than manually placing pixels, but various unexpected issues cast doubt on the choice in hindsight.

Choosing Java for its compatibility backfired. Not long before the project began, a new version of Java — Java 7 — was released, with stricter security requirements than previous versions, to the point of blocking older versions from running altogether. As a result, runnability depends on both the version of Java the applet is written in and the version of Java installed on the user’s computer, severely reducing the desired freedom from compatibility issues. Overall, getting file-accessing applets to run on modern versions of Java without paying for an official certificate proved difficult, and perhaps outright impossible if changing security settings or policy files on the clients is not an option.

As the lab room terminals used for developing the interface use a shared file system which is kept up to date, it was thought that there was no need to worry about the users’ Java version, but staff get their own personal computers and handle updating on their own.

Standard HTML was originally assumed to be insufficient for what the interface needed, but this seems to have been incorrect, given that there’s a HTML mode for GMail\(^4\). It is quite fast and lightweight, but more importantly, HTML is practically universally compatible. Then again, different browsers might still cause compatibility problems, and the GMail HTML mode was made by teams of professionals at Google, so attempting to mimic that as a lone student might not have been wise.

6.2 Accessibility, Usability, Acceptability, Engagement

These four factors provide a quick way to evaluate the design from the perspective of the user experience, with the initial results looking like so:

- Accessibility — Java Swing supports tools like screenreaders by default\(^5\) and should allow the interface to work on a variety of platforms — see section 6.1 for other thoughts behind the choice of Java. Sizes and shapes of interface components are also handled largely automatically, so different screen sizes such as with smartphones or reading pads ought to work out of the box. However, none of these special cases have actually been tested. Active support will be necessary if this interface goes through.

---

\(^3\)https://en.wikipedia.org/wiki/Swing_(Java)
\(^4\)https://mail.google.com/mail/?ui=html\(^5\)
\(^5\)http://docs.oracle.com/javase/tutorial/uiswing/misc/access.html
• Usability — There are always things that can be improved, see section [7.1] for some thoughts — but by and large the interface should be easy to use with a fairly low learning curve, mostly thanks to the straightforward functionality: select preferences, write text, see the results after clicking OK. All operations the interface can do are completed within half a second, which is effectively instantaneously to human brain, so there is no lag to speak of.

• Acceptability — Java Swing’s Look and Feel provides aesthetics fitting the user’s operating system, ensuring the visuals of the interface are generic at worst, professional at best; they ought to be acceptable in the expected academic setting. Functionality is less certain; again, section [7.1] details some issues and features left to future implementation, and the way the interface is fundamentally designed may give teachers too much control, which would be troublesome for the scheduling staff. This could be balanced out by more consistent data from the teachers, but it would require a full-scale attempt at adopting the interface to determine if it works out as net positive.

• Engagement — The weakest point for the interface. As mentioned above, Look and Feel ensures the aesthetics are those of the operating system, and as such it should mostly feel familiar rather than new and exciting. However, there is no sound or animation, and most of the interface is a flat grey tone, though this at least helps draw the eye to the parts that are coloured, like existing schedule requests.

6.3 Interviews

The initial interviews involved eleven individuals, two of them from the scheduling staff and the rest part of the teaching staff. It would have been useful to include higher administrative staff as well, as it turned out they had different needs from the other two groups, with the ability to get an overview of many different schedules being most important. On the other hand, it may be wise to leave that to a separate project, even if it uses the schedules produced with the help of this project, if only to avoid scope creep; see the discussion on future work, section [7.1].

As it had been decided to use a semi-structured approach for the interviews, the actual questions were roughly as follows:

1. Problems and conveniences in the current system, such as it is
2. Known similar systems (to the intended product)
3. Features the new system should have, issues it should avoid
4. Demonstration of how the scheduling process works currently for the interviewee
5. Preferred file formats, compatibility needs
6. Anything else
This approach was fine for the initial requirements gathering process, but trying to use it for the prototype evaluations was not as effective as the “think-aloud protocol” that was tried after the first set of evaluations. It is easier to determine what areas need work when observing a prototype in use than when asking users what they thought of the interface, though interviews may be necessary to get information on more abstract issues like familiarity (discussed in section 4.3.1).

6.4 Prototype

The earliest prototypes were born from the initial interviews and consisted of simple sketches in a notebook, as seen in fig 6.1. These sketches were discussed during the same interviews in regards to their potential features and issues, and soon had to be recreated digitally as the need to quickly and cleanly make changes in the design became pressing, as seen in fig 6.4. After the initial round of interviews were over, a functional prototype with some very rudimentary features such as pressable buttons followed, as seen in fig 6.8. The usual cycle of implementation followed, with staff testing out the prototype, changing the prototype according to the feedback, testing the resulting prototype, and so on. See the final result in fig 6.11.

6.4.1 Paper prototype

![Figure 6.1: Very first sketch of an idea](image-url)
Many things in this first sketch of an idea — fig[6.1] — remained all the way into the final version. There are also things that would have been nice to have but ultimately got skipped, as well as some indications of having misunderstood the task; notably, sessions being specific with times and places, rather than the as-broad-as-possible terms later versions would encourage. In no particular order:

- The little previews for the contents of other weeks (hard to see — it’s the little boxes under “v36”, “v37”, etc.). This made it into the final version, as it is very useful to be able to see a rough outline of other weeks at a glance.

- Using actual dates and weeks and etc. This was left out, as while it could probably have been accomplished in Java without too much trouble, the current abstract non-specific presentation has the advantage that it lets the scheduling staff decide when exactly the term starts for the teacher instead of being given a specific week for that, which is useful for them. It also makes reusing a requested schedule trivially easy, which is useful for the teachers.

- Ability to change language. Omitted entirely, which is not ideal if there are people at TekNat who are not fluent in English. This is unlikely to make the interface unusable for any of the intended users, as English fluency is widespread in academic contexts, but it could make it easier to use.

- Templates. The idea with the boxes on the left was that after creating a request for a session, you could save that request as a template and quickly make copies of the original request using said template. In this early stage this also extended to making templates of entire weeks or month so you could easily create a repeating schedule manually. Though it would be a pure quality of life functionality, it could be good to have in a future version; see section [7.1] for some discussion on the issue.

There are also some scrawls in the margins, ideas that came up during the interviews like indicating how many students or who exactly is attending — this was left to the scheduling staff or automated logic core, who presumably have that information already — or the strange “fun bonus: explosion sound effects whenever you click”, the reasoning for which is long since forgotten. A digitalized version of this design was attempted, as the sketch was rather messy; see fig[6.2] for the result.
6.4.2 Digitalization attempt 1

This design is too difficult to read, so using simple paint software for illustration was quickly abandoned. However, it does illustrate a feature that would have been useful: a visible link between a template and placed requests based on it. As changing the template was intended to also change the requests using it, this visible link should have made that connection more obvious, hopefully explaining the mechanic to the users intuitively. In hindsight, it might not have been enough, as there is no guarantee the user would have had a connected request visible when they changed a template, thus causing one or more changes to the schedule off-screen, which is all too likely to be missed and cause issues later.

Nevertheless, it would still be useful to be able to change an entire schedule at once in this way — perhaps the course code was changed, or labs would take place in a different location — so perhaps simply using a pop-up asking the user if they really want to change the template and listing what the effects would be could make the feature workable.
6.4.3 Paper prototype 2

![Paper prototype image](image)

Figure 6.3: Elaborating on the first sketch

The next sketch — fig. 6.3 — mostly went into more detail on things like how exactly to make a request, what are the options, etcetera. It also has more polish, such as the Uppsala University logo in the upper left corner, weeks in an orderly row, templates in a separate column to themselves, and so on.

The UU logo is partly to make the interface look more official and professional, but also space filler, making the main fields line up neatly without leaving a strange gap in the upper left. Previous versions had the “new request and templates” part of the interface up there, but it made the interface look too cluttered and busy.

On the right is the first instance of the “new request” form, at this point imagined as a pop-up. This is what users would fill out when they wanted to specify when they wanted a session, as well as whether it was a lab, lesson, etc; however, throughout the entire project, one of the biggest issues has been communicating to new users that requests should be kept as broad and non-specific as possible, to allow the scheduling staff to determine the details. The “earliest starting time” and “latest ending time” were intended to make this more intuitive, but even in the initial interviews it became clear this would not be enough. Ultimately, it might be necessary to write it out in plain text somewhere clearly visible, perhaps in a pop-up users have to click away before being able to use the interface.
6.4.4 Digitalization attempt 2

While the previous attempt at digitalizing the paper sketches had gone poorly, there was still a need for sketches that were cleaner in appearance yet still easily modified. Another attempt was made, this time using Dia\footnote{A free diagram creation program, also useful for mockups.} which worked out better and was used for all further mockups.

![Timetable preferences for: Week X of Year Y](#)

<table>
<thead>
<tr>
<th>Lessons &amp; Templates</th>
<th>Monday 25th Oct</th>
<th>Tuesday 30th Oct</th>
<th>Wednesday 31st Oct</th>
<th>Thursday 1st Nov</th>
<th>Friday 2nd Nov</th>
<th>Saturday 3rd Nov</th>
<th>Sunday 4th Nov</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lab-TDB</strong>;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 session(s);</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Between:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>08:00-17:00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feb 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 hours:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PL5**; C code:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TDB2K4F;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attending: Professor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teacherson, DVL, ITI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Comment: Not two in one day, preferably</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| **Busy**            |                |                  |                   |                 |               |                 |                |
| 1 session(s);       |                |                  |                   |                 |               |                 |                |
| 13:00-17:00         |                |                  |                   |                 |               |                 |                |
| Feb 17              |                |                  |                   |                 |               |                 |                |
| 4 hours long;       |                |                  |                   |                 |               |                 |                |
| (no location);      |                |                  |                   |                 |               |                 |                |
| (no c. code);       |                |                  |                   |                 |               |                 |                |
| Attending:          |                |                  |                   |                 |               |                 |                |

Figure 6.4: Dia "sketch" of a typical use case

Fig [6.4] is largely just a cleaner version of the previous attempt, but it has one new feature of particular interest: the “Busy” template. While not in the final version, it would be very easy to add, and would be used to allows users to specify when they are not available, as opposed to any other kind of request. With this, a single request could be used to cover a great length of time, while still allowing the user to mark days or times that must be kept free, providing some simple but flexible functionality.

Typically this prototype would be printed out and shown to interviewees, to make sketching changes and other notes quick and easy, while changing the design in Dia would be used for more lasting modifications, keeping the design clean. There were less major changes after this point, most notably the pop-up changing to separate the “standard” options from the more advanced ones, in the hope of encouraging teachers to be as unspecific as possible; see fig [6.5] fig [6.6] fig [6.7].
<table>
<thead>
<tr>
<th>Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of sessions:</td>
<td>Date:</td>
</tr>
<tr>
<td>Earliest starting time</td>
<td>Latest ending time:</td>
</tr>
<tr>
<td>Location:</td>
<td>Date:</td>
</tr>
<tr>
<td>Duration:</td>
<td></td>
</tr>
<tr>
<td>Course code:</td>
<td></td>
</tr>
<tr>
<td>Attending:</td>
<td></td>
</tr>
<tr>
<td>Comment:</td>
<td></td>
</tr>
<tr>
<td>□ Save as template</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Cancel</td>
</tr>
</tbody>
</table>

Figure 6.5: Early “New request” popup envisioned with Dia

| Number of sessions:          |                       |
| Starting date of block:      | Ending date of block: |
| Earliest starting time each day: | Latest ending time each day: |
| Duration:                    |                       |

(Required)

| Title:                           |                       |
| Course code:                     |                       |
| Location:                        |                       |
| Attending staff/student groups:  |                       |
| Comment:                         |                       |

(Optional)

| Only these weekdays:            |                       |
| Repeat block this many weeks:   |                       |
| Color:                           |                       |

(Advanced)

| □ Save as template    | OK                     |
|                       | Cancel                |

Figure 6.6: Later “New request” popup design
Figure 6.7: Last “New request” form design that used a popup approach, with tabs to ease navigation.
6.4.5 Proof of concept

![Figure 6.8: Skeletal prototype](image)

Figure 6.8 is the first prototype made with Java, and as such technically the first functional prototype. However, it had no response to input beyond buttons being depressed when clicked, and instead was used for showing what the layout and visuals might be like in functional prototypes, and discussing any changes that might be necessary even at this early stage.

In hindsight it was rather premature to do any testing at this point, as most of the interviewees were at a loss for what exactly to comment on. Discussing how useful it was to include Saturday and Sunday was about as far as it went.
6.4.6 Prototype 1

![First functional prototype](image)

Figure 6.9: First functional prototype

Fig 6.9 shows the first prototype that could be tested, while fig 6.10 shows the “New request” form, where users fill in what they want in their schedule; the popup approach was discarded in favor of making it a sort of new window, though it actually displays in the interface by hiding the normal schedule view.

Some needed changes were identified during testing:

- Already planned but not yet added functionality like changing or deleting existing requests, or identifying separate requests as referencing a single actual session.

- It needed to be clearer that the boxes along the top represented weeks and were clickable. Many testers tried clicking the boxes, but since nothing seemed to change — because every week was empty of requests, and therefore all looked the same — they never bothered with it again. There is a small indicator above the New request button showing what week is being viewed, but it clearly was not enough.

- Several testers requested a way of specifying the order of events even if they overlapped in possible times. This was implemented but ended up being very tricky to
explain, even to those who had requested that very feature; see the next prototype or the future work section for more discussion.

- The ability to click-and-drag to create requests would have been good, seeing as several users instinctively tried to do just that. However, due to fearing that it would take too long, this was never fully implemented, though the future prototype seen in fig[6.12] has some of this functionality.

Not shown in the images is that once a request is made, a small colored rectangle is added to the week view matching the settings of the request — the position reflecting when the request takes place, the color matching the type of session, and so on. This gives users a quick overview of the entire schedule made so far, and of course makes it much clearer what they are working with. Testers expressed appreciation the few times it came up; most took it for granted, as it is practically necessary and therefore standard for any sort of calendar-like view.

Figure 6.10: “New request” form for first prototype
6.4.7 Prototype 2

Fig [6.11] shows the main view of the interface. Without any requests made, there are few visible differences except for the number of days, but the week overview at the top has one. The week being viewed is indicated by a thick blue highlight, while a fainter highlight is given for the week the mouse cursor is hovering over, though the cursor itself is not in the screenshot. This was intended to make it more obvious to new users that the weeks are interactive, and new testers indeed tried switching week to a much greater extent than with the previous prototype.

Though it cannot be easily shown in a screenshot, the biggest change was the ability to double-click on requests to open them up for changing. This was indicated by a similar highlight when a request was clicked, but it turned out to still be too unclear that they were double-click-able, so it would make sense to instead only require a single click. It would also be more consistent, so this change seems obvious in hindsight.

In the request view (not shown) there are further changes: each request can now be given a unique session ID specified, granting the ability to have separate requests referring to the same session that was suggested in tests of the previous version. This is also when the "Happens-Before" field was added, which used the unique session ID to specify that two different requests with overlapping times had to happen in a specific order, though tests showed that it should probably have been "Happens-After" so users could create requests in order without having to go back and change previously made requests.

There are also smaller improvements, such as picking a type of session automatically selecting a default colour for that type, picking an earliest day ensures the latest day does not happen before it, and tooltips providing explanations for the fields. Ideally tooltips should not be needed, with context, names and visuals keeping features intuitively understandable, but coming up with something effectively self-explanatory for the "Happens-After" functionality proved too difficult.
### Figure 6.11: Second prototype
6.4.8 Unfinished future prototype

Figure 6.12: Unfinished prototype

There are many ways the interface could be further improved, but in the interest of finishing the project within a decade, the second prototype was chosen as an acceptable final version. But before that choice was made, some further development was attempted, with the unfinished results seen in fig 6.12.

As several tests brought up using the interface in a more graphical way, such as users selecting the starting time for a session by clicking and dragging instead of entering data by hand, some major restructuring of functionality was attempted. In this prototype, users can click on a given hour in a given day to start making a request matching that timeslot, with a teal highlight of the timeslot being hovered over making it clearer that it can be clicked. However this functionality was not critical for the interface to be usable, and was still buggy and unfinished, so instead of spending an unknown amount of time finishing it the previous version was used instead. It might still be useful as a visualization of the direction the interface ought to go; see section 7.1 for more thoughts on future development.
This concludes the evaluation of prototypes, but the development process itself also warrants discussion.

6.5 Testing methods

Testing was done using the "think-aloud protocol". Section 4.4 goes over it in more detail, but essentially it works by testers using the product and speaking their thoughts aloud, whereas the test conductor observes and makes notes while trying to keep assistance at a minimum.

Testers were given a set of tasks to try and complete using the prototypes. Originally the plan was to rapidly iterate by making changes to the interface between each test according to the feedback, so the very first prototype — the proof of concept in figure 6.8 — lacked functionality. However, this would make the feedback inconsistent, making it difficult to prioritize feedback accurately, so this approach was abandoned just after testing began. As a result the first round of testing was somewhat awkward, as two tasks were impossible due to the needed functionality not being implemented, so testers were simply told how it would look and act or shown with drawings, similar to the paper prototypes during the initial interviews. Finishing the functionality or adjusting the tasks would have been appropriate.

The original set of tasks went as such:

1. Create requests covering several weeks, details not important
2. Save a schedule, make changes, load old save
3. Remove requests
4. Delete the saved schedule

Where tasks 1 and 3 were impossible due to unimplemented functionality. 4 usually also got skipped as it was already clear from task 2 that the saved schedule could be manipulated like any other file, which was the main intent.

For the tests of the next prototype these tasks were changed to better reflect existing functionality as well as what staff might use the finished interface for:

1. "You want to request a fairly typical week, with a lecture at some point in the first three days, then a lab in the last two."
2. "You want a lecture in one week followed by a lesson and a lab in the next, but you’re not picky about which day they happen, just that the lab happens after the lecture."
3. "You change your mind; the lesson should be a lecture, and the lab can also happen in the same week as the first lesson."
4. "Tired teachers and students are no good, so you want a lesson no earlier than ten in the morning, and a lab after lunch in the same week, with a note to the scheduling staff that they should happen on the same day."

66
The resulting feedback was noticeably more thorough, so using this style for the tasks is definitely preferable.

An issue with the think-aloud method arose, at least as it was used in this project. Testers would turn to the test conductor to ask for clarification almost immediately if anything was unclear; while this is useful as a sign that something could be made clearer, it also means any issue that would only require a bit of time to solve in a real usage scenario cause the same reaction as issues that could not have been solved so easily.

One solution that could perhaps have brought the testing sessions closer to a real usage environment would be recording testers with a camera (with their consent obviously) while the test conductor stays in another room altogether. However, even in this case testers might be affected by the presence of the camera — in wanting to express how something is problematic to the test conductor, they might turn to the camera and explain their problem. As explaining a problem is a common way to figure out how to solve it, even this much would make testing conditions different from real use cases.

6.6 Personal challenges

Here the author would like to acknowledge personal problems noticed throughout the project, in hopes of helping others avoid my mistakes. This section will be more informal due to the personal and subjective nature of the content.

The first problem was that I picked the project out of a list of potential thesis works that TekNat wanted students to take on, not because it sounded interesting, but because it seemed within my comfort zone. It’s a common problem for anyone with a “fixed mindset” [Carol Dweck, 2014], where you feel that any failure is a sign of fundamental inadequacy rather than just something that needs more work. Anyone who was consistently praised for qualities they don’t have any control over instead of the work they put in is likely to develop this mindset; if you’re called “smart” for acing a test, then it won’t feel as if the studies you did beforehand were the deciding factor, so you’re not incentivized to study in the future. What’s worse, if you then fail at a test, logically it must be because you’re stupid, even if no parent or teacher would ever say that.

Because of this mindset I have consistently had issues with trying on things I wasn’t already good at, as well as keeping up work on anything that just needs time and effort, both of which are terrible for when you need to finish your education by writing a thesis. Moreover, due to feeling anxiety about my skills not being enough for the project, I’d put off working on it, which increased the anxiety I felt whenever I thought about working, though this didn’t get really bad until after I’d been quite sick and thus legitimately unable to work for a week. After that whole week of not working, the anxiety was enough to make me not want to think about the issue and thus put it off even more, entering a negative spiral.

Chronic procrastination is hardly a new problem though, so there was help from a variety of sources, including Studentthålsan, an organization with psychologists specializing in issues students often have. There I got the “five-minute method”, which helps you start working despite anxiety by setting a short timer before you start, then when it goes off deciding whether you can stomach working some more. [Abigail Billingslea, 2012] describes it under the heading of “easing into the task”. Doing a small amount of work
like “write for five minutes” is much easier than something big and abstract like “work on the project”, and by getting any work at all done, the anxiety should be lessened next time. This works best if combined with a regular schedule, such as setting an alarm for a given hour every day and working for however long you can manage each time.

Nevertheless, it took time to even decide to seek help, in part because of not wanting to admit that it was needed — after all, one of the core problems was the fear of being deficient somehow, so the thought of admitting help was necessary caused some anxiety of its own. If I hadn’t found out about the fixed mindset problem by chance it might have taken even longer.

Perhaps this could all have been avoided by not deciding to work alone. At the time, I both didn’t know anyone with an interest in HCI, but also wanted to be independent. I had recently worked together with someone on a project and found myself slacking, which got me worried that my work ethic wasn’t as good as I had thought. A justified worry as it turned out, but I also missed out on useful criticism and feedback, particularly outside of the testing phases. Moreover, it’s considered wise in HCI to have a third person around during interviews to keep an eye on the reactions of interviewer and interviewee, especially what they say, as well as take clear notes throughout.

In any case, working on the project has had a big effect on me as a person. My issues are hardly gone, but it now feels realistic to think they might be at some point.
Chapter 7

Discussion

Most testers expressed generally positive sentiments about the overall approach of the project during prototype testing, so even if it did not fully automate the scheduling process it should provide us with some understanding of what the staff want out of a fully automated scheduling setup. Still, it is recommended to use some existing commercial scheduling interfaces, perhaps one from section 2 or at least get an entire team of people working on the interface rather than just the one student.

On a broader level, this project could be said to have contributed to the field with the interface itself despite all its problems, as well as by narrowing down what is wanted of a scheduling interface. Most valuable is perhaps an understanding of the problems the interface had in previous versions and those it still has, as they inform any future efforts at automating scheduling. Whether by making a new interface, modifying the one from this project, or picking one of the commercially available choices.

7.1 Future work

Perhaps the most important functionality deliberately left out was translating the produced schedules into form usable by the existing automated scheduling core [Hellkvist, Sjostedt 2012]. Given that the output files have a consistent serialized format, it should be possible to convert them to suit the logic core with a script, but some internal code changes may be needed.

Using version control for the entire project rather than just the source code seems extremely obvious in hindsight, as it ended up being worked on over multiple computers, so changes to the file system would be missed when trying to just copy over the main folder contents. Not to mention getting an overview of the progress made via the commit messages.

Security went ignored, as the program is only run by users on their own computer without making any connections to the outside, so as long as internally produced Java software is considered trustworthy there should be no problems.

Thread-safety similarly got little attention as multi-thread functionality was never added. The interface is in no way computationally intensive, so forcing it to run on a multiple threads seemed unnecessary. But if applets use threads automatically by default, thread safety would need more attention; the documentation of the libraries used, such as Java Swing, frequently mentioned that many features were not thread-safe.
Higher-up administrative staff could find it useful to view many or even all finished schedules, but then they would most likely want to have some separate viewing functionality to aid this, with features like filter by teacher, course code, and so on. The interface was not made with this functionality in mind, but if it becomes needed badly enough it should be possible to do.

7.1.1 Specific features

Known issues and desired features:


2. Something like the template functionality, perhaps the ability to designate a week as a default week, so any week that is left empty is treated as if it had the contents of the designated week, making repeating schedules much easier to set up.

3. Session type “Away” or “Busy”, to allow a chunk of time to be marked as an exception to an overlapping request.

4. Tweak colors — more professional, and easier to read text, but keep the high contrast for the sake of colorblind users.

5. Make Session ID and Happens-Before (or -After) clearer, as they generally had to be explained directly to testers.

6. Allow specifying a week a request is made in — or even multiple weeks — manually, rather than the roundabout method currently used where you select the week you want the request to be in by clicking on it in the week mini-preview view.

7. Make it clear when an existing request is being modified, rather than a new one being created. Currently these use the exact same screen.

8. The note telling users that “more specific requests are less likely to be accepted” could be clearer in intent and visuals.

9. For Session ID, show which IDs are already in use, to avoid mistakes and also to ease using the Happens-Whatever feature.

10. Stop “new request form” from remembering the contents of previous requests by default, as this tended to do more harm than good. Possible exception for the course code field, which is likely to stay the same.

11. Make the contents of the “comments to staff” field show up more clearly, as they are likely to be high priority. Possibly even display it on a separate part of the interface.

12. Allow arrow keys or some other keyboard command to change week, and indicate this is possible somehow.
13. Accept/Delete/etc buttons were a bit hard to spot, and hardly any tester noticed that Delete became clickable while modifying an existing request, so these could perhaps be moved or colored to stand out.

14. Only handle normally scheduled times of day like 8-17.

15. Some graphical interactivity, like clicking on a given hour to create a request starting that hour.

16. The rectangles representing Requests in the main view currently take up less space when there is more than one in that same day so they can be seen alongside each other, but this does not take into account the time of day, which it should. Or use a different approach altogether.

Some of these were already added to the unfinished future prototype in fig 6.12 but even then most were not finished. In conclusion, the produced interface basically does what it was intended to, but could be improved dramatically. However, this project will not make further changes to it.
Bibliography

Johan Ludde Lundin. Discussion and walkthrough of feasibility study.

Henning Hellkvist and William Sjöstedt. Toward Automated Timetabling at TekNat.


Phillip Kevin Reed. A Comparison of Programming Languages for Graphical User Interface Programming.


Jacqueline M. Tappan, David J. Pitman, Mary L. Cummings, Denis Miglianico. Display Requirements for an Interactive Rail Scheduling Display.

Catherine Plaisant, Ben Shneiderman, Jim Battaglia. Scheduling Home Control Devices: A case study of the transition from the research project to a product.


Nick De Veil. Personas Considered Harmful.

KronoX — framtidens schemaläggningssystem (the timetabling system of the future). [http://www.kronox.se/](http://www.kronox.se/)

A. Sasha Giacoppo. The Role of Theory in HCI.

J. Nielsen. Thinking Aloud: The #1 Usability Tool.

StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share.


W3Schools. Internet Explorer Browser.


Kara Pernice. Talking with Users in a Usability Test.


Carol Dweck. TEDxNorrkoping: The power of believing that you can improve.

The Anti-Procrastination Workbook. ISBN: 9781105747267