Design and implementation of a web-based time tracking system

Ludvig Widman

October 6, 2011
Master’s Thesis in Interaction Technology and Design-Engineering, 30 credits
Supervisor at CS-UmU: Thomas Johansson
Examiner: Pedher Johansson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

In this exam project a time tracking system is designed, evaluated and implemented at the Umeå based company Codemill AB. The system used within the company at the start of the project had several usability problems and lacked important features. The goal of the project was therefore to create a system that better meets their needs and can easily be extended with new features when necessary.

A requirements study collected information about what the users needed, how other systems had solved similar problems and what the issues with the existing solution were. A prototype was created from the requirements and evaluated with inspection methods and user studies.

The refined prototype was implemented as a web application. The final system was also evaluated with user studies and refined based on feedback.

Design and implementation of a web-based time tracking system
# Contents

1 Introduction ............................................. 1

2 Problem Description .................................... 3
   2.1 Problem Statement ................................... 3
   2.2 Goals .................................................. 3
   2.3 Purposes .............................................. 3
   2.4 Methods .............................................. 4

3 Requirements study ..................................... 5
   3.1 Existing systems ..................................... 5
      3.1.1 Kimai ............................................ 5
      3.1.2 Harvest .......................................... 6
      3.1.3 Project Hamster .................................. 6
      3.1.4 Toggl ............................................. 6
      3.1.5 Tick ............................................... 9
   3.2 Contextual Inquiry ................................... 9
   3.3 User Study ........................................... 9
   3.4 Results from User Study .............................. 11
      3.4.1 Subjects ......................................... 11
   3.5 Evaluation of User Study .............................. 13
   3.6 Gathered requirements ............................... 14

4 Prototype ............................................... 15
   4.0.1 Track time ......................................... 15
   4.0.2 Week view ......................................... 15
   4.0.3 Reports ............................................. 15
   4.0.4 Settings ............................................ 17
   4.1 The design process .................................. 17
   4.2 Prototype evaluation ................................. 17
      4.2.1 Cognitive Walkthrough .......................... 17
      4.2.2 Heuristic Evaluation ............................. 18
4.2.3 User tests ................................................. 20

5 Implementation .............................................. 23
   5.1 System overview ........................................ 23
   5.2 The Server and Django .................................. 23
      5.2.1 How Django works ................................ 24
   5.3 The web client .......................................... 26
      5.3.1 The parts of the interface ......................... 27
   5.4 REST API ................................................ 27
      5.4.1 REST ................................................. 28
   5.5 Unit tests ................................................ 29

6 Evaluation .................................................. 31
   6.1 Results from user tests ................................. 31
      6.1.1 Bugs and other problems ......................... 31
      6.1.2 Requested features .............................. 32

7 Conclusions .................................................. 35
   7.1 Achievement of goals and requirements ............. 35
   7.2 Restrictions ............................................ 35
   7.3 Limitations ............................................. 36
   7.4 Future work ............................................. 36

8 Acknowledgements .......................................... 37

References ..................................................... 39

A Web client GUI ............................................... 41

B API Documentation .......................................... 45
   B.1 Access .................................................. 45
   B.2 Listing all elements of a type ....................... 45
   B.3 Details for one element .............................. 46
   B.4 Modifying tracks ...................................... 46
   B.5 Other methods for tracks ............................. 47
      B.5.1 Stop ............................................... 47
      B.5.2 Stop all .......................................... 47
      B.5.3 Start (or Create) ................................ 48
Chapter 1

Introduction

This is an exam project for the Master of Science Programme in Interaction Technology and Design-Engineering by Ludvig Widman. The exam project was done at Codemill AB in Umeå. The goal of the project was to create a new time tracking system for Codemill, to be used internally and to be sold externally.

The design of the new system was based on an user study of their current system and an overview of other systems.

A prototype was created, evaluated with heuristics, revised, tested with users and once again revised.

The prototype was then implemented as a real web application which was tested by users and iteratively improved upon.
Chapter 2

Problem Description

2.1 Problem Statement

The problems this project intends to solve are all related to the current time tracking system, Kimai, and the flaws associated with it.

One of the main problems with Kimai is the lack of reporting features and integrates poorly with economy systems. This results in large amounts of extra work in order to generate invoices and handle salary payments.

Another problem is that the interface has bugs and usability problems. The interface is especially cumbersome for users that prefer to report time retroactively.

2.2 Goals

The ultimate goal of this project is to build a system that can replace Kimai and solve the issues associated with it. To accomplish this a series of sub-goals must first be completed:

- A requirements study must be carried out in order to determine what the users need in a time tracking system and how the issues associated with Kimai could be solved
- A prototype needs to be build and evaluated
- The system must be implemented as a web application that can replace Kimai and must have an Application Programming Interface (API) that can be used to create additional software that talks with the system
- The resulting system needs to be evaluated with user tests

2.3 Purposes

The purpose of the project is to optimize the time reporting routines within Codemill AB by constructing a purpose-built system that avoids the pitfalls of the current implementation.
A secondary purpose is to construct a generic and extendable system that retains its value regardless of the implementation context, thus enabling Codemill AB to use it as a revenue stream via licensing or software sales.

2.4 Methods

This project will use context appropriate methods for the different stages of the project.

In the requirements study the method Contextual Inquiry was utilized to study how the current system was used. Researching other systems and how they have solved similar issues is another method that was used in this phase.

In the prototype phase several evaluation methods were used. First the prototype will be evaluated by the report author to find the most obvious issues. For this the methods Heuristic Evaluation and Cognitive Walkthrough were utilized. User tests were then used to determine what issues the users had with the prototype.

The implementation phase included unit tests to make sure that the system worked correctly and user tests to see that no issues were introduced during the implementation.
Chapter 3

Requirements study

The goal of the requirements study was to find out how the new system should be designed. This was accomplished by studying existing systems, the users of the current system and other potential users of the new system. For studying users the method Contextual Inquiry was used.

3.1 Existing systems

A lot of time tracking systems exist today. Wikipedia lists over 30 notable ones [5]. It would be impractical to study all of them so a selection of five popular or otherwise interesting systems was made.

The selected time tracking systems are described and analysed, with a focus on how time is reported. The primary goals with this study was to get different perspectives on how time can be tracked and to find useful features to include in the system. The studied systems will not be tested for how good they are as time tracking systems in a more general sense.

3.1.1 Kimai

Kimai [16] is a open source web based time tracking system, and also the system Codemill used at the start of this project. In Kimai time is reported by selecting a customer, project and task and then pressing the start button. Previously reported activities are shown in a list and can be used to quick start the same type of entry. The reported activities can also be edited and a comment can be added. The interface is shown in Figure 3.1 on the following page.

A date picker at the top of the interface lets the user select for which period the previously reported activities should be shown. A sum of the reported time during the specified period is also shown. It’s not possible to select a date in the future as the end date. Once a range is selected it does not automatically change at midnight, so the user has to manually change the range every day.
Figure 3.1: Time tracking in Kimai is done by selecting the type of entry and pushing the start button. The small start buttons next to each previous item can also be used to quick start that type of item.

In an administrative view users, clients, projects and tasks can be added, edited and removed. This view is not shown for regular users, only for administrators.

3.1.2 Harvest

Harvest[13] is a commercial web based time tracking solution that also includes invoicing and tracking of other expenses.

In Harvest time reporting is done through a time sheet with both day and week views. In day mode all tasks for one day is shown with corresponding timers that can be started and stopped. In week mode the total number of hours per day for each type of task can be edited. This is shown in Figure 3.2 on the next page.

3.1.3 Project Hamster

Project Hamster[20] is an open source time tracking software for Linux. It has some unique features. One of these is the ability to specify when a new day should begin to avoid breaking up work over midnight. Another interesting feature is to switch task automatically when switching to a specific workspace.

In Project Hamster time is reported by specifying the current activity, as shown in Figure 3.3 on page 8. By starting a new activity the previous one are stopped. Activities can also be stopped manually. Activities can be grouped in categories and tagged. Reports can be made on different activities, categories and tags.

3.1.4 Toggl

Toggl [24] is a commercial web based time tracking system. Time is tracked in Toggl by entering the current task and pushing start. Editable properties include project, customer, user specified tags and whether the task is billable. Previously reported activities are shown in a list and can be edited. The interface is shown in Figure 3.4 on page 8.
Figure 3.2: In Harvest time is reported through a time sheet. Here it’s shown in both day view (top) and week view (bottom). In day view the button New Entry is pushed to show how new tasks can be added.
Figure 3.3: In Project Hamster time is reported by switching between different activities. Tags can also be attached to each activity.

Figure 3.4: In Toggl time is tracked in a start/stop fashion. Entries can be tagged and marked as billable.
3.1.5 Tick

Tick [23] is a commercial web based time reporting system. Time is reported by filling out a time card, shown in Figure 3.5 on the following page. The time card has fields for client, project, task, amount of time and notes. This system lets the user specify a time budget for each project. A progress bar displays how much is left of each time budget.

3.2 Contextual Inquiry

Contextual Inquiry is a qualitative data gathering method where the subjects are observed and interviewed while they perform their work. The method is based on three principles: The study takes place in the subjects real work environment, the subject and the researcher cooperate on equal terms to explore the issues and the study is based on areas of interest rather than specific questions [21].

The idea behind Contextual Inquiry is that when observing users within the context of their work they can easily enter into a conversation about the what, how and why of their work. The researcher and subject can together discover things about the system the subject only knew implicitly [14].

Users are much better at contributing with their real work experience if they are interviewed in the context of that work than if they are invited to a design meeting [14].

Since Contextual Inquiry can take a substantial amount of time per subject, perhaps as much as a half day, it is often used with purposive sampling. That means that the subjects are selected based on the purpose of the study rather than randomly among users of the system [21]. The Raven 1996 paper (see ref [21]) suggests visiting three subjects in each market and that these three should be as different from each other as possible. The motivation for this is to be able to find both differences and overlap with a small amount of subjects.

If the activity that is of interest for the study takes place sporadically over time a variant of Contextual Inquiry called Artifact Walkthrough can be used. With this method the subjects are asked in advance to prepare for the study by documenting their use of the activity. The subjects are then asked to recreate the documented usage during the study [21].

Contextual Inquiry uses focus areas rather than specific questions. To get answers for specific close-ended questions it is common to conduct a Survey as a compliment to the Contextual Inquiry.

Some common methods for analysing the data from Contextual Inquiries are data affinity diagrams and work flow diagrams [14] [21]. Data affinity diagrams group items of data into related areas with representative headings. Work flow diagrams represents the roles users take on when using the system and how different roles coordinate and communicate [14].

3.3 User Study

Contextual Inquiry was used as the method for the user study. It included subjects primarily from Codemill but also from external companies.
Figure 3.5: In Tick time is tracked by filling out a time card. A progress bar shows how much of the time budget is used.
3.4 Results from User Study

Notes were taken and audio was recorded during the study to enable later analysis of the interviews.

The study contained three focus areas, each with several questions:

1. How time tracking should work
   - Describe your ideal way of reporting time?
   - Do you prefer start/stop or filling out afterwards?
   - How would you like to switch between tasks?
   - How would you like to group time tracking (projects, tasks, clients, categories, tags)?
   - Would you appreciate the ability to see a time budget for the current project?
   - How many different tasks do you switch between on a day?

2. How the current system is used
   - What do you think about Kimai?
   - Is there anything you are missing in Kimai?
   - What problems have you encountered while using Kimai?
   - How do you use Kimai?
   - How do you use comments in Kimai?

3. How reports should work
   - What kind of reports do you need from a time tracking system?
   - How do you follow up time utilisation in projects?

The idea was not to ask each question to each participant but rather that the questions should form a base for discussion related to the focus areas.

3.4 Results from User Study

3.4.1 Subjects

All of the following subjects except Maria work at Codemill. The subjects works as developers or with administrative tasks. They were chosen to include as many different types of users as possible.

Daniel

Daniel is a developer and uses Kimai to report time on different projects. He usually reports time for about four projects during normal a week. He states that he prefers to use the start/stop feature in Kimai instead of reporting afterwards. However, he says that he dislikes that he has to stop a task before he can write a note about it.
He mentions that sometimes he needs to add activities afterwards instead of using start/stop. He usually uses the history in Kimai to add the correct type of activities and then edits them afterwards. He would appreciate a proper feature for favourites as a complement to this.

Another problem he mentioned with Kimai is the time interval chooser. It requires the user to select the interval to be shown each day otherwise newly added activities just disappears from the view. The user also has to refresh the browser to be able to select the new interval.

He requests integration with the ticket system so that he can see tickets in the time tracking system and start reporting time for them directly. Another alternative we discuss is to make it possible to start time tracking from the ticket system.

Another feature he would like to have is a time budget on a ticket basis so that an estimate can be added to each ticket and then compared to the actual time it took. He would prefer to also have a graphical representation of this so it's easy to see how much of the time budget is left.

Daniel also proposed adding a “suggest project” feature, so that the users can suggest a new project and start reporting time on it. The administrators could then be notified and either approve the new project or merge it with some existing project. We discussed the importance of informing the user if the suggested project is merged with another project, as it would be very confusing if the project just disappeared.

When I ask him about a mobile client he says that he sees no need for one for himself but other might appreciate it.

**Jon**

This subject works both as a sub consultant and a normal developer. Jon states that he does not like Kimai and therefore uses another way to report time. He sends weekly time reports by email where he just fills out the day and the amount of hours on each project. His reasons for not using Kimai is that it takes to much time to report time. In order to fill out the number of hours for a project he needs to create a new row, fill out a start time and a stop time and select client, project and task.

When he works as a sub consultant he has to use the system at the clients workplace. He prefers the system one client he has worked for uses over Kimai but says that he only used a fraction of the features in it.

Jon would like to see the amount of vacation days he has left for the year in a time tracking system. He would also appreciate the ability to see flextime (how much he has worked so far in comparison to how much he has to work).

**Rickard**

Rickard mostly uses the administrative side of Kimai. He adds new clients and their projects, adds new users and generates reports for invoices.

In the administrative view in Kimai he can add clients, projects, tasks, users and groups. Clients and projects has a one-to-many relation and tasks are global to all projects. Rickard would appreciate the ability to adjust which tasks should be available for which project.
3.5. Evaluation of User Study

He also mentions that he has a relative that would like to have a many-to-many relation between clients and projects, and that this feature is missing in most time reporting systems.

To generate reports for invoices and salaries he uses an external script that connects to the Kimai database and returns the desired data. This is not something that can be done from the interface in Kimai. He expressed a desire for improved integration between the time tracking system and the economy system, since the current data transfer methods relies on manual workarounds.

Another improvement he would like to see is the ability to report vacation and sick days in the time tracking system. He would also like to be able to input the expected work hours per month. This would make it possible for the system to show each worker how much flextime they have or if they are behind their expected amount of work.

Ingrid

Ingrid works with administrative tasks like salaries and invoices so she only uses the reports from Kimai. She would like to see some better follow up features, like reports for how accurate time estimates was. She also requested features for vacation days and sick days in the time tracking system.

She suggested that different users could benefit from having different tasks in Kimai when reporting time. Perhaps non used tasks should be auto hidden after a while. Another alternative might be to specify tasks based on user group or let the users decide for themselves.

Maria

Maria works at Uminova Innovation and we discussed how they report time and her thoughts on time tracking in general.

She currently uses a Excel based system with a time sheet that is filled in and submitted each month. The time sheet has one column for each day and rows for each project. The number of hours per day on each project is filled in as well as any time off.

She is of the oppinion that reporting time afterwards is less stressful than using a start/stop system. She also says that it would be best to support both methods; flexibility is important, since different users have different needs.

Good reports that are easy to customize is something she would appreciate in a time tracking system. She wants to be able to look at the data from different perspectives to be able to learn and improve the way she spends her time.

3.5 Evaluation of User Study

Contextual Inquiry has worked well for understanding how a system is used by its users. All of the inquiries have been interesting and given good insight to the needs of the users. It seems like people like talking about how they work. The conversations flows easily and relatively little input from the researcher has been necessary.
Since the users are studied within their real work environment the problems that are detected are very real. The users frequently has opinions regarding the problems, including suggestions related to improvements.

Another benefit of studying users while they work with the system is the possibility to directly discuss solutions to some problems.

### 3.6 Gathered requirements

This is a summary of the requirements gathered for the new time tracking system. The requirements are ordered according to the MoSCow Method [17].

**The system must have:**

- Support for both the use of a start/stop timer and reporting time afterwards.
- A way to specify Client, Project and Task for each Activity.
- Some functionality for reporting Activities as overtime.
- History for previously reported Activities. Items in the list must be editable and possible to delete.
- Reports for at least: amount of reported hours per employee and month, amount of reported hours per project and month, reported vacation and overtime.
- A feature for reporting vacation, sick days and other time off.

**The system should have if possible:**

- An indicator of how much time there is left to work the current month.
- A way to set ticket ID for some Activities
- A time budget indicator for projects, Activities and tickets. This would also require a way to set the time budget.
- The ability to select which tasks that should be visible per user.

**The system could have if it does not affect anything else:**

- The ability to suggest new projects that can be approved or merged by an administrator.
Chapter 4

Prototype

A prototype was created based on the requirements specified earlier. Some of the screens from the prototype can be seen in Figure 4.1 on the next page. In the top part of the prototype is the main navigation. It lets the user choose between: Track time, Week view, Reports and Settings. To the right in the top is a summary of how much the user has worked during the current month and how much is left. The different tabs are described in detail below.

4.0.1 Track time

This view is where time can be tracked in a start/stop fashion. To the left is an area where the current Activity can be entered and below is a history of recently tracked activities. To the right is an area that can be customised by the user. In this configuration three favourites and a ticket list are shown. Favourites are frequently tracked projects the user can switch to very easily. The ticket list is an integration with a ticket system that lets the user start tracking time for one of the tickets.

4.0.2 Week view

In this view the user can report their time afterwards on a day basis. The view shows an entire week at a time and the user can fill in how many hours they worked on different projects during each day. If a project is missing it can be added via the Add Row button.

4.0.3 Reports

This view lets the user view reports generated from the time tracking data. A set of pre-designed reports will be available to the user to choose from.
Figure 4.1: Two of the views from the first prototype. The topmost picture shows the time tracking view with three favourites and ticket integration added in the right column. The picture below shows the week view with four projects added.
4.1 The design process

The prototype was initially sketched on paper paper, and the design later transferred to Apple Keynote. The paper sketches was used to determine a very coarse layout for the different parts and the finer details were added in the digital sketches.

The sketches was based on the gathered requirements, guidelines [25][6][19] and many years of experience designing websites.

4.2 Prototype evaluation

The prototype was first evaluated by the report author with two inspection techniques called Cognitive Walkthrough and Heuristic Evaluation. Subsequently user tests were carried out. One benefit of using inspection techniques first is that obvious problems can be corrected so that the user tests can yield better results [6].

Normally one would use a group of experts to do these types of evaluations, preferably experts in both Human Computer Interaction (HCI) and the domain of the evaluated software [18]. In this project the author has done the evaluations on his own.

4.2.1 Cognitive Walkthrough

Cognitive Walkthrough is a inspection method that focuses on how easy a software is to learn by exploring its features. The motivation for this is that its more efficient if the users can learn by using the software instead of investing time in formal training and that many users prefer to learn new systems by trial-and-error [2].

The method was developed in the 90s because it was too difficult to apply the theoretical models of the time in real projects. Many theoretical models required building a complete simulation of the design which could be as complex as building the proposed application itself [3].

The method is based on Cognitive Theory and involves simulating the users cognitive activities to ensure that it’s easy to learn and perform the tasks the system should support [4].

Cognitive Walkthrough primarily focuses on ease of learning, which is only one aspect of usability. It is therefore recommended that this method is used in combination with other evaluation methods [2].

To use this method, a prototype of the interface is needed as well as some specific user tasks that should be performed in the prototype. For each task an action sequence of actions
required to complete the task is created. The action sequences is then walked through in
the prototype and each action is evaluated [2][4].

For each action, the following questions is evaluated:

– Is the correct action evident for the user?
– Can the user connect the description of the action with what they intended to do?
– Can the user determine if a correct or wrong choice of action was made based on the
  systems response?

The following tasks were evaluated within the prototype.

1. Tracking time for a project and writing a note about the Activity
2. Reporting time for work on three different projects during the previous two days
3. Getting a report for how much overtime I have worked the last month
4. Editing a previously reported Activity to have a different task

Results from the evaluation

The first task did not rise any issues.

The second task requires an indicator that informs the user that the filled out values are
saved. It would also be beneficial to have a summary of the number of hours reported per
day so the user can check if it adds up to the expected amount.

The third task had no issues but a link to a list of reported activities with overtime could
be useful. Similar links could also be added for other types of reported time, e.g. time off.

The fourth task needs some way of saving an edited Activity so that new activities can be
entered. This is because the same part of the interface is used for both adding and editing
activities.

4.2.2 Heuristic Evaluation

Heuristic Evaluation is an inspection method where an interface is evaluated against a list
of commonly accepted guidelines and principles – heuristics. It is suitable to use early in the
development process since it can detect problems with just prototypes. Correcting problems
is usually much easier in a prototype stage before anything is implemented.

Early lists of heuristics was quite long. This made them time consuming to use and ineffi-
cient. The motivation for using heuristics instead of users is to save time, so a short list
makes more sense. A new list with only the most important heuristics was therefor proposed
by Nielsen in 1994 [18]. This list is the one used in this study and it’s included in Table 4.1.

The evaluation was made on each screen in the prototype.
### Heuristics for Interface Evaluation

<table>
<thead>
<tr>
<th>Heuristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visibility of system status</td>
<td>The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.</td>
</tr>
<tr>
<td>Match between system and the real world</td>
<td>The system should speak the users’ language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.</td>
</tr>
<tr>
<td>User control and freedom</td>
<td>Users often choose system functions by mistake and will need a clearly marked “emergency exit” to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.</td>
</tr>
<tr>
<td>Consistency and standards</td>
<td>Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.</td>
</tr>
<tr>
<td>Error prevention</td>
<td>Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.</td>
</tr>
<tr>
<td>Recognition rather than recall</td>
<td>Minimise the user’s memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.</td>
</tr>
<tr>
<td>Flexibility and efficiency of use</td>
<td>Accelerators – unseen by the novice user – may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.</td>
</tr>
<tr>
<td>Aesthetic and minimalist design</td>
<td>Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.</td>
</tr>
<tr>
<td>Help users recognise, diagnose, and recover from errors</td>
<td>Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.</td>
</tr>
<tr>
<td>Help and documentation</td>
<td>Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user’s task, list concrete steps to be carried out, and not be too large.</td>
</tr>
</tbody>
</table>

Table 4.1: Heuristics for interface evaluation from Nielsen (1994) [18].
Results from the evaluation

In the track time screen the stop buttons could be made a different colour when pressed to better show the system status. The ID number for each Ticket could be removed, especially since each ticket links to the corresponding page in the ticket system. In the display in the top right corner the Expected field could be removed since it’s not necessary and makes other information less visible.

The week view could be improved by making saving more apparent. An indicator that shows when the changes has been saved or an explicit save button are two possible solutions. The addition of week numbers could help some users. The hour format for the total column should be the same as in the rest of the prototype. Some faster method of navigating to a week further away might be useful for some users. The total for each row is not really necessary, it would be more useful with a column total.

The reports screen had very few issues. It would be beneficial to add links from certain reports to more details reports regarding the same information. Another issue was that there was no way to edit or remove reports.

The settings screen requires a save button. Some of the settings pages could also be merged.

4.2.3 User tests

After the problems found with the inspection methods were resolved the prototype was tested with real users. All the users in the tests work at Codemill.

Jon

Jon was positive towards the prototype, especially the week view. He suggested an integration of time estimation in the week view so that users can estimate how much they will work on each project each day in advance and then approve or edit the time afterwards.

The time budget meters needs some improvement to make it more apparent which meter shows what. Its unclear if any given meter shows the time budget for the project, an Activity in the project or for a ticket.

Dennis

Dennis suggested a change to a "1:30" time format instead of the current "1h 30min".

Another suggestion was a more compact history that only takes up one line per Activity instead of the current two. An alternative could be to make this a setting so the user can decide.

Rickard

Rickard thought it was good to have both a week view and a start/stop view. He suggested that the expected amount of hours was shown in the week view to make it easier to compare the totals.
4.2. Prototype evaluation

He liked how the reports view works with a set of custom made report the user can choose from. He also agrees that time budgets should be possible to set on a project, task and ticket basis.
Chapter 5

Implementation

5.1 System overview

The system consists of four big components: the server, the web application, the database and other clients. Of these the server and the web application are the result of this project.

The server is written in python using the web framework Django, which will be described in section 5.2.

The server builds the web application which runs in the users browser. It also provides a Representational State Transfer (REST) API for other clients. The data in the system is stored in a database but all communication goes through the server. This is shown in Figure 5.1 on the next page.

5.2 The Server and Django

As stated earlier, the server is based on Django [7]. Django is a high level web framework written in Python. It provides a structure that separates data processing, logic and presentation into Models, Views and Templates which makes it a MVC framework. Django is based on the DRY principle [11] which means that it minimises repetition in the code by automating as much as possible.

Django was chosen for this project for several reasons. It’s a promising and popular framework that seems suitable for the project. Django makes it easy to develop web applications in an efficient manner. Another reason is that Django is Python based, which is a language the author already knows, which places focus on project performance rather than technological education. Django also has a nice open source license: the BSD license, which lets you do more or less anything with it.
5.2.1 How Django works

Django works by receiving Requests from a client and processes it to a Response which it
sends back. The requests are first sent to the URL Router which matches the requested
URL to a View. The View gathers data from one or several Models and prepares it for use
in a Template. The Template is then rendered and sent back as a Response. This chain is
shown in Figure 5.2 on the facing page.

Models

The Models describes the types of data used in the application and is used as an abstraction
of the database. They also provide high level methods for processing the data [1].

Each model usually corresponds to a table in the database. The model has fields of different
data types that match columns in the table. Each instance of the model is a row in the
table [8].

The models hides the interaction with the database so that instead of writing SQL the
developer works with the model objects. Methods for retrieving and filtering objects is
available through the Model Manager [8].

Relationships between models uses a ForeignKey type that also does reverse look-ups au-
tomatically, for example: if a Project has a Client then the Project object can access the
Client object and the Client object can get a list of Project objects that references to it.

Some of the more important models used in this project is Client, Project, Task, Track,
Report and User.
5.2. The Server and Django

Views

A View handles Requests sent to one or several URLs. Most views gathers data from one or several Models and then prepares it for use in a Template. Usually each view corresponds to a specific template [1].

The view contains whatever logic is necessary to return the response. The response does not have to be a template, it can be a redirect, an error code, a file or anything else that would make sense as a response [10].

Templates

A Template is basically a document with insertion points for data. It can be a XML-file, a HTML document, or any other text based format. In this document special template code is inserted to indicate where different pieces of data should be included [9].

Templates have some limited scripting possibilities such as loops and if-statements. Filters for formatting data in different ways is also available [9].

A template can extend or include other templates. This makes it possible to avoid repetition by creating hierarchies of templates that extends eachother. A template can define blocks of content. These blocks can then be altered in a child template [9].

In this project most templates are HTML documents for some segment of the application. A base template is used for the general layout, this is extended by templates for each view that includes other templates for specific components of the view.

Figure 5.2: The server receives a Request from a client. The Request is routed to the correct View. The view fetches data via the Model and renders a Response from the data and a Template.
5.3 The web client

The Graphical User Interface (GUI) of the web client consists of four views: Track time, Week view, Reports and Settings. A description of each view follows:

**Track time**

The Track time view allow the user to track time by starting and stopping a clock. In this view time is tracked on an activity basis, meaning that each time a clock is started a new activity is reported. A history shows previously reported activities. Favourites gives the user easy access to the most commonly used projects.

**Week view**

This view allows the user report time afterwards for each day of the selected week. The user adds a row for each project it has worked on and can then report time for each day in a grid of projects and days.
5.4. REST API

Reports

The reports view allows the user to analyse reported time in different ways. A selection of reports are available, including an employee summary a project summary and an overview report.

Settings

The settings view allows users with sufficient privileges administer the system. This might include adding new projects, editing available tasks, updating the working hours and adding new users. Regular users can also use this view to change their password and update their profile.

5.3.1 The parts of the interface

Each part of the interface in the web client is shown in figure 5.3 on the preceding page. Detailed screen shots of each view can be found in Appendix A on page 41. A description of each part follows:

1. **Time tracker** Let’s the user start and stop a clock for an Activity. Client, project, task and overtime can be selected. A note about the Activity can also be written.

2. **History** Shows previously tracked activities. Client, project, task, note, time span and duration is shown. Activities can also be edited and restarted.

3. **Favourites** The user can add often used combinations of client, project and task here for easy access.

4. **Week view** In this view time can be reported for each project and task for each day of the week.

5. **Add rows** New project-task combinations can be added with this part.

6. **Report selector** The user can select which report to display here.

7. **Report** This is where the report is displayed.

8. **Time span selector** The time span for the report can be changed here.

9. **Settings selector** The user can select which settings panel to display here.

10. **Settings** The selected settings panel is displayed here.

5.4 REST API

The system has a API that can be used by other clients to communicate with the server. This API is designed to follow the REST principles described below.
5.4.1 REST

Representational State Transfer (REST) is a style of software architecture for server/client systems communicating over Hypertext Transfer Protocol (HTTP). REST architectures have a set of constraints they must adhere to:

- Clients are separated from servers by a decoupled interface. Clients should not be concerned with server tasks such as data storage and servers should not be concerned with client tasks like the user interface. The servers does not even keep track of which clients it have talked to, so if a URL changes the clients that depend on it will not know until they try to access it [15].

- The client-server interface is uniform. This means that a fixed set of HTTP verbs [22] (e.g. GET, POST, PUT, DELETE, etc) are used to access and modify any type of resource. It does not matter if the resource is an XML file or a HTML document [15].

- The client-server communication is stateless. This means that each request should contain all the information required to understand it, regardless of previous requests [12].

- Responses to GET requests are normally cacheable, which means that they do not modify any data and should return the same response for every request (until the data is modified). Responses should state for how long they can be cached, if at all [12].

- A client can not tell if it is communicating with an end server or an intermediary. This makes load balancing and caching easier to implement since an intermediary server is just as good as an end server for many requests [12].

If a system conforms to the REST constraints it can be referred to as being RESTful.

5.4.2 The API

The API can be used to list and modify objects in the system. API calls must be authenticated with a valid user name and password from the system. Since the API calls are for one user only they can only view and edit what a normal user can.

The following object types can be listed and viewed with the API:

- projects
- clients
- tasks
- tracks
- overtime
- favourites

Of the above object types, currently only Tracks can be modified.

All API calls is in the form of HTTP Requests to a specific API URL with either the GET or POST verbs. The API consumes GET-encoded POST data and returns objects as XML.
5.5. Unit tests

A example of an API call that views a specific project follows:

GET /api/projects/5/
<?xml version="1.0"?>
<Project>
   <Id>1</Id>
   <Name>Internt</Name>
   <Client>1</Client>
   <Colour></Colour>
   <TotalTime>3 days, 0:01:22</TotalTime>
</Project>

The first line is the call and the following lines are the response that is returned.
A more detailed explanation of how the API can be used is included in Appendix B.

5.5 Unit tests

Unit tests were used to avoid bugs in the more error prone parts of the system. For example, the report that is used to transfer data to the economy system was heavily tested.

Every part of the system was not tested with unit tests. Some parts of the system was not very suitable for unit tests, like the user interface. Other parts where to trivial to gain much from automated tests.
Chapter 6

Evaluation

The web client was evaluated by letting a small group of users from Codemill use it for an extended period of time. The evaluation started with just one test user which was extended to four users after the first week.

The users were instructed to use the system to track their time and to report any issues they encountered. Suggestions for new features or other improvements was also encouraged.

Development of the system was continued during the evaluation. Bugs were fixed and new features were added when the users requested them. This made the testing both dynamic and iterative as the users were testing newer and newer versions of the system.

6.1 Results from user tests

The user tests resulted in a lot of bug reports, issues and feature requests from the users. The results are grouped in two categories: Bugs and Feature requests.

6.1.1 Bugs and other problems

Several bugs were found during testing. An overview of the type of bugs that were most commonly found follows.

Bugs with multiple users

Several bugs related to multiple active users were found early in the user tests. These bugs were fixed quite early in the testing.

Bugs in error handling

A few problems with how the system handled bad input were discovered. These were often related to how date and time was formatted. Some of the reports also had some issues when
no data was available. These bugs were fixes as they where found during the testing.

**Broken buttons**

Some interface buttons stopped working during the testing phase, due to newly added features that interfered with the button code.

**Interface issues**

A number of minor interface issues were also found. This includes clocks that didn’t tick, broken lines due to long names and charts with too many items.

### 6.1.2 Requested features

A lot of features were requested during testing. Some features were implemented while others were not. This selection was based on the perceived usefulness and the estimated implementation time of the feature. A selection of implemented requests follows.

**Transfer to economy system**

Since all reported time at Codemill has to be entered in the economy system an early request was a report that simplified this procedure. A report that outputs the exact same data that the economy system requests was built to meet this need.

**Overtime**

Support for reporting overtime of different types was requested. The possibility to select the type of compensation for reported overtime was also asked for. These features were implemented in a overtime-selector in the track time view where both type and compensation can be selected in a drop down menu.

**Tasks in Week View**

The tested version of the Week View only displayed projects, not tasks. Since the economy system does not support entering time for a project without also selecting a task this had to be changed. To meet this need the Week View was altered to display project/task combinations instead. With this change support for activities without a task selected was dropped throughout the system.

**Improved reports**

Several improvements of the reports were suggested and implemented. A new graph for different work types (Normal work, overtime, time off, etc) over time was introduced. Labels
6.1. Results from user tests

in charts were truncated if they were too long. Items in the charts was also ordered by total time reported.

Other features

Several minor feature requests were also implemented. Start and stop time were added to the history, support for different input formats in week view was added, an indicator for gaps in the history was added and the favourites were improved in several ways.
Chapter 7

Conclusions

7.1 Achievement of goals and requirements

The overall goal of the project was to build a system that could replace Kimai and solve the issues associated with it. Several sub goals were set to reach this goal. A summary of how these sub goals were fulfilled follows:

- A requirements study focusing on user-needs and the problems with Kimai was carried out
- Prototypes were created and evaluated
- The system was implemented as a web application
- The web application was evaluated with user tests

It is my opinion that the system developed during this project is good enough to replace Kimai. The system also solves the issues that Kimai has.

The system meets all the must-have requirements (see Section 3.6 on page 14) and some of the should-have requirements. The requirements that have not been met have been moved to the Future work section below.

7.2 Restrictions

The application currently only supports one group of users (e.g. one company that uses the application). It is not possible to have different groups of users that sees different clients, projects, tasks and so on. This restriction could be avoided either by adding an extra layer of grouping for everything or by deploying separate instances of the application for different user groups.

The interface was developed with the assumption that users do not want to enter seconds. The application still tracks seconds internally, but time is presented and entered without them. For example, if a clock is stopped, the seconds are recorded, but when shown in history the time is rounded to whole minutes.
7.3 Limitations

Projects are limited to having exactly one client each. One client can have several projects but not the other way round. This makes it easier to handle projects as the client can be implied (e.g. when reporting an activity only the project has to be entered, not the client).

When data is transferred to the economy system some precision is lost due to rounding. This limitation derives from the fact that the economy system handles time as a decimal number (i.e. 1.5 hours instead of 1:30). Decimal hours cannot represent certain fractions precisely. For example 20 minutes cannot be represented with a fixed number of decimals.

When tracking time over midnight the time is reported on the day the Activity was started instead of being split across both days.

7.4 Future work

Several interesting features could be implemented in the future.

Integration with a ticket system could improve time tracking for users that mostly work on tickets. This feature was included in the prototype but later removed since it would require disproportionate amounts of work for full-scale implementation.

Transfer to the economy system could be improved. Currently it’s a report that can easily be transferred. This could instead be made fully automatic so that no manual step would be required.

Tracking of vacation days, overtime compensation and travel expenses is all relevant and could be added to the system.

Time budgets for projects or users could be a useful feature. If time budgets should be set per project / month, per project in total, per user, per goal in a project or in some other way must be evaluated. Perhaps several different ways of setting time budgets must be supported.

A feature for suggesting new projects could be very useful for users that work on new projects that have not yet been added to the system.

A customisation that some users might appreciate is the ability to select which tasks that should be visible on a user basis.

Other possibilities include integration with other economy systems, included invoicing features, integration with project management software and other similar features.
Chapter 8

Acknowledgements

I would like to thank my internal supervisor Thomas Johansson for his support and help with the report and the process for the project.

I would also like to thank my external supervisor Ingrid Westman for her help and support with the project.

All the people I interviewed and performed user tests with should also receive a special thanks, this project would not have been possible without you.

I would like to thank Anna Wadenstein and Alfred IV West who helped me with the report and contributed their opinions on the prototypes.
References


Appendix A

Web client GUI

Figure A.1: Track Time: In this view time can be tracked using a clock. A history display shows previous entries. To the right several favourites can be added for easy access.
Figure A.2: Week view: Time is reported day by day for each project/task combination. A sum of all entries during each day is displayed at the bottom.
Figure A.3: Reports: The user can select from several types of reports. The report at the top shows the reported time for one user during a month. The report at the bottom shows each day during a week for each project.
Figure A.4: Settings: If the user has sufficient privileges they can administer the system from the settings view. Topmost is a view for editing working hours during a year. At the bottom is the Django admin interface where new projects and clients can be added etc.
Appendix B

API Documentation

B.1 Access

To use the API you must first be authenticated. Currently the standard authentication in the time tracking system is used. If you are logged in you should be able to access the API.

B.2 Listing all elements of a type

Objects can be listed with the following call:

GET/api/<object_type>/

Where <object_type> can be any of the following:

- projects
- clients
- tasks
- tracks
- overtime
- favourites

Example:

GET/api/project/

Response:

<?xml version="1.0"?>
<Projects>
  <Project>
    <Id>1</Id>
  </Project>
</Projects>
<VerboseName>Codemill: Internt</VerboseName>
</Project>
<Project>
    <Id>2</Id>
    <VerboseName>Company: Other project</VerboseName>
</Project>
</Projects>

B.3 Details for one element

To view more than just the ID for an element, use the more detailed call:

GET/api/<object_type>/<id>/

Example:

GET/api/project/1/

Response:

<?xml version="1.0"?>
<Project>
    <Id>1</Id>
    <Name>Internt</Name>
    <Client>1</Client>
    <Colour></Colour>
    <TotalTime>3 days, 0:01:47</TotalTime>
</Project>

B.4 Modifying tracks

Track objects can be modified with the following call:

POST /api/track/<id>/
<data>

Where <data> should be GET-encoded parameters like:

project=1&amp;task=3

The available parameters are:
- project
- task
- start
- stop
- note
B.5. Other methods for tracks

- overtime
- delete

Where project, task and overtime should be IDs. Project, task and start are required. This request will return the updated track object or a HTTP error 500 with a message. If delete is specified the track will be deleted and a 200 OK is sent.

Example

POST /api/track/1/
project=1

Response:

<?xml version="1.0"?>
<Track>
   <Id>1</Id>
   <Project>1</Project>
   <Task>1</Task>
   <Start>2011-02-28 13:17:00</Start>
   <Stop>2011-02-28 14:17:01</Stop>
   <Note>Test test</Note>
   <Overtime></Overtime>
   <Colour>green</Colour>
   <Duration>1:00:01</Duration>
</Track>

B.5 Other methods for tracks

There are several methods that can be used to modify tracks.

B.5.1 Stop

Stops a track that has not yet been stopped.

POST/api/track/<id>/stop/

Returns a redirect to the track object if successful, otherwise one of the following errors:

<table>
<thead>
<tr>
<th>Error code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>403</td>
<td>You are not the associated user for this track.</td>
</tr>
<tr>
<td>404</td>
<td>The specified track does not exist.</td>
</tr>
<tr>
<td>500</td>
<td>The track has already been stopped or request method was not POST.</td>
</tr>
</tbody>
</table>

B.5.2 Stop all

Stops all running tracks for the logged in user.
POST/api/track/stop_all/
Returns Error 500 if POST is not used, otherwise a 200 OK.

B.5.3 Start (or Create)

New tracks can be started with the start command. This is currently the only way to create a track.

POST /api/track/start/
<data>

Where <data> is GET-encoded parameters. At least Project and Task must be specified. Overtime and Note can optionally be specified. If successful a redirect to the new track object is returned, if not Error 500 is returned.