Calendar 2.0
The mitigation of calendar friction
Abstract

Calendar applications have been a primary tool for time management, although they have been slowly developed and improved. The slow development of calendar applications has created friction of usage.

In this report, we propose a friction mitigation strategy for calendar applications. A set of characteristics are identified: active, intelligent, personal, extensible, information-centred and dynamic view, which describes the notion of friction.

The goal is to develop a framework, supporting the prototype and providing reusable components for future development. The prototype is evaluated by participants and the result is used to conclude that friction can be mitigated to some extent.

Keywords: mitigation of calendar friction, active calendar application, frictionless calendar.
Abstrakt

Kalenderapplikationer är, och har varit, ett primärt verktyg för tidshantering även om de har utvecklats och förbättrats långsamt. Den långsamma utvecklingen av kalenderapplikationer har skapat en friktion vid användning.

I den här rapporten föreslår vi en strategi för att mildra friktionen som skapas av kalenderapplikationer. En uppsättning av egenskaper identifieras: aktiv, intelligent, personlig, extensible (tänjbar), informations centrerad och dynamisk vy, vilka beskriver begreppet friktion.

Målet är att utveckla ett ramverk som supportar prototypen och innehåller återanvändbara komponenter för framtida utveckling. Prototypen utvärderas av användare och resultatet används för att dra slutsatsen att friktionen kan mildras i viss utsträckning.

Nyckelord: Mildring av kalenderfriktion, aktiv kalenderapplikation, friktionsfri kalender.
Contents
1. Introduction ................................................................. - 1 -
  1.1 Background ............................................................ - 1 -
    1.1.1 Physical calendars .............................................. - 1 -
    1.1.2 Calendars as a desktop/web application .................. - 1 -
    1.1.3 Calendars as a mobile application (App) ............... - 2 -
    1.1.4 The existing calendar applications ....................... - 2 -
  1.2 Problem ...................................................................... - 3 -
  1.3 Purpose and Goal ...................................................... - 4 -
  1.4 Restrictions .................................................................. - 5 -
  1.5 Structure of Report ..................................................... - 5 -
2. Theory .............................................................................. - 6 -
  2.1 Ongoing work ............................................................. - 6 -
    2.1.1 The existing personal assistant applications ........ - 6 -
    2.1.2 Other existing applications: Meeting notes .......... - 6 -
  2.2 The Rational Unified Process ........................................ - 7 -
  2.3 Software prototyping .................................................. - 8 -
  2.4 Software evaluation .................................................... - 9 -
    2.4.1 Participants ........................................................... - 9 -
    2.4.2 Tasks ..................................................................... - 9 -
    2.4.3 Test data ............................................................... - 9 -
    2.4.4 Summary of test data ............................................. - 10 -
3. Methods .......................................................................... - 11 -
  3.1 Tools ........................................................................... - 11 -
  3.2 Setup ........................................................................... - 11 -
  3.3 Requirement analysis .................................................. - 11 -
  3.4 Design and architecture .............................................. - 12 -
  3.5 Framework and technology ......................................... - 13 -
  3.6 Third-party libraries and mechanisms ......................... - 14 -
  3.7 Implementation .......................................................... - 14 -
    3.7.1 Foundation ............................................................. - 14 -
    3.7.2 CRUD operations .................................................. - 14 -
    3.7.3 View navigation ..................................................... - 15 -
    3.7.4 Event type and user specific properties ............... - 15 -
    3.7.5 Active events ....................................................... - 16 -
    3.7.6 Fine tuning ........................................................... - 16 -
    3.7.7 Testing ................................................................. - 16 -
Appendix C: Evaluation form

Appendix B: Guidelines for future development

Appendix A: Requirements

References

4. Evaluation results ........................................................................................................... - 17 -

4.1 Task 1: Week navigation ......................................................................................... - 17 -

4.2 Task 2: One week planning ..................................................................................... - 18 -

4.3 Task 3: More advanced, two week planning ............................................................ - 18 -

4.4 Task 4: User specified properties ........................................................................... - 19 -

4.5 Task 5: Active Event ............................................................................................... - 20 -

4.6 Answers to general questions .................................................................................. - 20 -

5. Discussion .................................................................................................................... - 22 -

5.1 Evaluation result and research question .................................................................. - 22 -

5.2 Discussion of the chosen methods .......................................................................... - 23 -

5.3 Project positioning .................................................................................................. - 24 -

5.3.1 Previous work ...................................................................................................... - 24 -

5.3.2 Virtual personal assistants .................................................................................. - 25 -

5.3.3 Meeting notes applications ................................................................................... - 25 -

5.3.4 Limitations of findings ........................................................................................ - 26 -

5.4 Future work .............................................................................................................. - 26 -

6. Summary and Conclusion .......................................................................................... - 27 -

Appendix A: Requirements ............................................................................................ - 30 -

Requirement list .............................................................................................................. - 30 -

Scenarios ....................................................................................................................... - 32 -

Appendix B: Guidelines for future development ......................................................... - 34 -

Class diagram ................................................................................................................ - 34 -

Component diagram ...................................................................................................... - 35 -

Framework ..................................................................................................................... - 35 -

Prototype ....................................................................................................................... - 38 -

Test ............................................................................................................................... - 41 -

Sequence diagrams ...................................................................................................... - 41 -

Appendix C: Evaluation form ....................................................................................... - 44 -

Task 1: Week navigation ............................................................................................... - 45 -

Task 2: One week planning ........................................................................................... - 45 -

Task 3: More advanced, two week planning ................................................................. - 47 -

Task 4: User specified properties ................................................................................ - 47 -

Task 5: Active Event ..................................................................................................... - 48 -

General questions ......................................................................................................... - 49 -
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.1: Microsoft Calendar</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2.2: The process of software prototyping</td>
<td>8</td>
</tr>
<tr>
<td>Figure 3.1: The logical view (Class Diagram)</td>
<td>12</td>
</tr>
<tr>
<td>Figure 3.2: The development view (Component Diagram)</td>
<td>13</td>
</tr>
<tr>
<td>Figure 3.3: The structure of the project solution</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3.4: The event form</td>
<td>15</td>
</tr>
<tr>
<td>Figure 4.1: Navigation paths for task 1</td>
<td>17</td>
</tr>
<tr>
<td>Figure 4.2: Navigation paths for task 2</td>
<td>18</td>
</tr>
<tr>
<td>Figure 4.3: Navigation paths for task 3</td>
<td>19</td>
</tr>
<tr>
<td>Figure 4.4: Representation of conflicted events</td>
<td>21</td>
</tr>
<tr>
<td>Figure 5.1: A mitigation strategy</td>
<td>23</td>
</tr>
<tr>
<td>Figure 5.2: The layer of calendar application</td>
<td>25</td>
</tr>
<tr>
<td>Figure B.1: Calendar 2.0 class diagram</td>
<td>34</td>
</tr>
<tr>
<td>Figure B.2: Calendar 2.0 component diagram for the framework</td>
<td>35</td>
</tr>
<tr>
<td>Figure B.3: The structure of the framework project</td>
<td>35</td>
</tr>
<tr>
<td>Figure B.4: An example of PreferenceUtility usage in Controller</td>
<td>36</td>
</tr>
<tr>
<td>Figure B.5: An example of Hibernate subclass usage in Events.hbm.xml</td>
<td>37</td>
</tr>
<tr>
<td>Figure B.6: An example of Hibernate session usage in GEventUtility</td>
<td>37</td>
</tr>
<tr>
<td>Figure B.7: The structure of the prototype project</td>
<td>38</td>
</tr>
<tr>
<td>Figure B.8: An example of week view for 2013 in Index.aspx</td>
<td>39</td>
</tr>
<tr>
<td>Figure B.9: An example of authentication in LoginController</td>
<td>40</td>
</tr>
<tr>
<td>Figure B.10: Representation of conflicting events</td>
<td>40</td>
</tr>
<tr>
<td>Figure B.11: The structure of the test projects</td>
<td>41</td>
</tr>
<tr>
<td>Figure B.12: Manage an event attached to external service</td>
<td>42</td>
</tr>
<tr>
<td>Figure B.13: Get notifications for active events</td>
<td>42</td>
</tr>
<tr>
<td>Figure B.14: Manage an event</td>
<td>43</td>
</tr>
</tbody>
</table>

List of Tables

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1.1: The brief comparisons of features in different calendar applications</td>
<td>3</td>
</tr>
<tr>
<td>Table 2.1: A comparison between Sherpa and Google Calendar</td>
<td>6</td>
</tr>
</tbody>
</table>
### List of abbreviations

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calendar 1.0</td>
<td>Existing calendar applications.</td>
</tr>
<tr>
<td>Calendar 2.0</td>
<td>A new layer on top of Calendar 1.0.</td>
</tr>
<tr>
<td>Calendar friction</td>
<td>A comprehensive description of the obstacles of usage in existing calendar applications</td>
</tr>
<tr>
<td>All-day event</td>
<td>Events occur during a day, and the time is unknown. Such events can last one or more days.</td>
</tr>
<tr>
<td>Whole-day event</td>
<td>Events occur from 00:00 to 23:59 the same day.</td>
</tr>
<tr>
<td>Cross-days event</td>
<td>Events start at one specific time in a day, and end at specific time in another day.</td>
</tr>
<tr>
<td>Normal event</td>
<td>Events start at one specific time and end at another specific time in the same day.</td>
</tr>
<tr>
<td>Read-only event</td>
<td>Non-editable events from subscribed calendars.</td>
</tr>
<tr>
<td>Active event</td>
<td>Events associated with an external service.</td>
</tr>
<tr>
<td>Event type specific properties</td>
<td>Properties specific for a type of event, not part of a superclass.</td>
</tr>
<tr>
<td>User specific properties</td>
<td>Fields that can be specified by users for any event. The fields exist for all event types and are sometimes referred to as customised fields.</td>
</tr>
<tr>
<td>User specific property names</td>
<td>Names users can attach to each user specified property.</td>
</tr>
<tr>
<td>Event type</td>
<td>Type of event defining a set of fields serving a specific purpose, e.g. GLecturerEvent.</td>
</tr>
<tr>
<td>External service</td>
<td>A source of information that resides outside the scope of calendar application, e.g. weather forecast, train/flight information, etc.</td>
</tr>
<tr>
<td>Calendar</td>
<td>A group of events in the context of calendar application.</td>
</tr>
<tr>
<td>CRUD</td>
<td>Basic data operations: Create, Read, Update, and Delete.</td>
</tr>
<tr>
<td>SRS</td>
<td>Software Requirements Specifications</td>
</tr>
</tbody>
</table>
1. Introduction
Calendars are for many people an obvious tool. Due to the considerable use of computers and mobile devices in the last few decades, there is an increasing interest in calendar applications. The development of digitised calendars has progressed slowly, whilst other applications have been developing rapidly.

Calendars serve as the primary source of time-related information, helping users organise their time. Features have been added to make the calendar more proactive and intelligent. However, calendar applications are still rather trivial.

By looking back at previous calendar applications, it becomes clear that few features have been added. Moreover, searching other topical applications on the market shows several attempts to make up for the shortcomings of calendar applications.

1.1 Background
"How do we keep track of time?" (Van Helden, 1995) is a question asked by many people in the history of humanity. In the beginning, the calendar was used to keep track of tasks, i.e. when to plant crops and when different religious ceremonies were to take place (Van Helden, 1995).

The background covers how calendars are developed from physical calendars to the calendar software applications. It specially emphasises the features of existing calendar applications.

1.1.1 Physical calendars
Physical calendars were invented as a tool for managing events since the origin of calendar. By time, the complexity of calendar increased, and the representation on paper was not enough to fulfil all requirements. Whilst information has a requirement of being digitised, the physical calendars are also required to be developed as a software application.

1.1.2 Calendars as a desktop/web application
As the information technology rises, computer is very capable of organising information. Calendar was considered to be digitally made. The noticeable improvement is that a reminder feature was introduced to calendar applications. In 1985, Microsoft developed an application named Calendar (Wikipedia, 2013a) superseded by Microsoft Schedule Plus. As can be seen in the following figure, the application has an alarm function. According to our research, that application might be the very first software program for calendars (Wikipedia, 2013a). In 1996, Microsoft Schedule Plus was retired (Wikipedia, 2012b); however the main functionality was introduced to Microsoft Outlook 97. And since then, most calendar applications are intended to be developed along with other services such as email. For instance, Lightning in Thunderbird, provided scheduling functionality, was developed by Mozilla Foundation in 2004 (Wikipedia, 2013c). Google Calendar, which is a web application of personal organiser, was released in 2006 (Wikipedia, 2013d).
1.1.3 Calendars as a mobile application (App)
In addition, digital calendar appears in the field of mobile device. A digital calendar in a mobile phone usually has a reminder feature. Since smartphone has been developed, calendar is already a mobile application as a basic function in a smartphone. Moreover, calendar apps nowadays have a mechanism of synchronisation with a user’s account. Google Mobile Calendar (Google Play, 2012a) and Microsoft Mobile Outlook (Google Play, 2012b) are a very good example to support event synchronisation. That is, users modify event’s information in a smartphone, and the information can be shown in different devices.

1.1.4 The existing calendar applications
As the number of users of Microsoft Outlook and Google Calendar increases, it appears that the features from those applications have become the foundation of calendar application. Other calendar applications have been also developed by adopting or extending features from those such as Yahoo Calendar. The following table shows the brief comparisons of features in different calendar applications, i.e. Microsoft Outlook Calendar, Google Calendar and Yahoo Calendar.
### Microsoft | Google | Yahoo
--- | --- | ---
**Multiple Calendars** | ✔️ | ✔️ | ✔️
**Views**
- Day
- Work Week
- Week
- Month
- Schedule View | Day
- Week
- Month
- 4-Days
- Agenda | Day
- Week
- Month
- Year
- List
**Reminder Types**
- Alarm
- Reminder Window
- Email
- SMS
- Messenger (MSN) | Reminder Window
- Email
- SMS | Alarm
- Reminder Window
- Messenger
**Priority Support** | ✔️ | ✗ | ✗
**Import/Export** | ✔️ | ✔️ | ✔️
**Calendar Sharing** | ✔️ | ✔️ | ✔️

**NOTE:**
- Multiple Calendars: several calendars for grouping events
- Views: different ways to present events in the screen
- Reminder Types: different options for reminder
- Priority Support: A feature indicating the importance of events
- Import/Export: Import or Export event’s information
- Calendar Sharing: A feature that allows users to share calendars

| Table 1.1: The brief comparisons of features in different calendar applications |

### 1.2 Problem
Calendar applications have been developed by different organisations or companies for nearly two decades. Very few new features are yet introduced. Calendar applications could have been a more advanced tool of time and information management. Thus, there is room for significant improvement in such applications.

As can be seen in Table 1.1, most calendar applications share a number of similarities. Some expected features are discovered due to people’s behaviours. However, no matter what new applications are created, the features do not correspond with the expectations to some extent. This mismatch causes friction, i.e. a gap between the existing and expected features in a calendar application.

The friction of calendar application in this context can be described by the deficiency of some or all following characteristics: intelligent, active, personal, extensible, information-centred, and dynamic view.
Intelligent
The existing calendar applications provide little support for helping users with the planning process. An intelligent calendar application should provide assistance in scheduling for users in terms of logical operations, better usability and less uncertainty.

Active
Events are frequently changed for different reasons, which users of calendar applications have to manually apply changes for. An active calendar application should provide support for automatic updates when changes occur. Moreover, it should actively interact with users instead of forcing them to interact with it. This interaction removes uncertainty and reduces their workload.

Personal
A personal calendar application should allow users to personalise events. It should provide different types of events with different user and event type specific properties. Therefore, users should be able to create their own personal view of the calendar.

Information-centred
An information-centred calendar should indicate information according to time and serve as a central application of information. Thus, it should provide an efficient tool for information management.

Dynamic View
Our research shown in Section 1.4 indicates that existing calendar applications have provided four or five static views for users. It indeed displays information in different views, but the views are not consistent.

As mentioned above, calendar applications cause friction of usage, which decreases the software quality. This leads up to the following questions:

Why does the friction exist? Can it be mitigated? If the answer is positive, what impact will that have on the usage?

1.3 Purpose and Goal
In general, friction created by existing calendar applications restricts the software quality. By mitigating the friction, calendar applications can provide better functionality for users and increase the value of the software itself. The purpose of this project is to investigate if friction can be mitigated by addressing some of the characteristics and how this will affect the value of calendar applications.

The goals are threefold:

- to establish a framework as the foundation and
- to specify and implement a set of requirements to build a prototype calendar application and
- to evaluate the prototype in regard to friction

Development will be achieved in two steps. Firstly, a set of requirements for the framework and the prototype application to be created should be identified. Secondly, a prototype application should be built based on the following aspects:

- the requirements collected and
- the deficiency of characteristics as mentioned previously

1.4 Restrictions
There are a considerable number of calendar services in use, which makes it impossible to include all in this research. Three commonly used calendar services are considered regarding the functionality: Google calendar, Yahoo calendar and Outlook calendar. Google calendar gives most attention to the project team. For the sake of simplicity for this project, some irrelevant features of calendar applications are not considered unless they will have an explicit impact on the evaluation outcome. For example, security is not considered as a major concern in the project.

Some calendar related applications are taken into account. However, those will not be considered as the core source for the requirements in this project. Again, too many applications exist to be able to investigate them all. Hence, the main focus of the project is the calendar applications as mentioned above.

Some restrictions are applied to the prototype such as resolution and web browser support. Those are critical since the prototype is to be built as a web application. According to statistics of web browsers in use from W3C School, Chrome and Firefox are the most commonly used browsers (W3C Schools, 2013a). Based on their statistics, the prototype application is to be tested and executed in those browsers. W3C School (2013b) also states that the most common resolution is above 1024 by 768. More precisely, TechCrunch (2012a) suggests that the most popular resolution is now 1366 by 768.

In addition, there are some restrictions on the framework. Since the libraries are implemented using C#, a .NET language is required for their usage. The current integration only supports the Google calendar service and its trivial operations.

1.5 Structure of Report
The subsequent sections in this report consist of theory, methods, evaluation results, discussion, summary and conclusion, as well as references and appendices. Theory section elaborates the theories considered as a guideline such as software prototyping. Methods section explains the critical methods applied in the project. Evaluation results section elicits comments on the prototype from participants. Discussion section provides responses to our problem based on our evaluation results, as well as information on reasoning of the chosen methods, positioning of findings and future work. Summary and conclusion section describes our conclusion on the project. At the end of the report, there is information on material and literatures in References section. In addition, extensive information on the project can be found in the Appendices section.
2. Theory
This chapter elaborates theories considered as our project guidelines. It covers ongoing work, software engineering principles, and techniques for software evaluation. Those theories provide significant insights into our project.

2.1 Ongoing work
In this section, calendar-related applications which are ongoing improvement based on calendar applications are briefly summarised. It focuses on two types of applications: personal assistant application and meeting note application.

2.1.1 The existing personal assistant applications
The use of virtual personal assistant apps dramatically increases these days as a large number of different apps have appeared in the market. One of them is called Sherpa, which has between 100, 000 and 500, 000 installations (Google Play, 2013c). Sherpa is one of the most popular apps on Android regarding virtual personal assistants (TechCrunch, 2013b). Sherpa quickly provides answers to the given statements. As an example, it plays songs based on statements made by users. In the video on TechCrunch (2013b), Sherpa is asked to play a song by Green Day. Sherpa displays a playlist with Green Day songs, downloads, and plays one of them instantly.

Another virtual personal assistant app, Donna which is available as an IOS app, is also popular. Donna proactively collects information in the background to help making decisions in day-to-day tasks. As an example, it collects transport information (e.g. how long it will take you to park?) to calculate the estimated time to travel between two locations (TechCrunch, 2013c). The following table shows a comparison between Sherpa and Google Calendar features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Sherpa</th>
<th>Google Calendar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proactive</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Play songs on command</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Add appointments to calendar</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Weather forecast</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 2.1: A comparison between Sherpa and Google Calendar

Moreover, Google has developed a personal assistant named Google Now. It is a mobile phone application that provides information requested from different web services and displays it in “cards” format (Wikipedia, 2013f). Google Now also has the ability to predict information that users may need based on their actions on the application (Wikipedia, 2013f).

2.1.2 Other existing applications: Meeting notes
Retrace was recently released at Disrupt NY’s Startup Battlefield. Retrace is an app that assists users in organising meetings by adding information on top of Google calendar. Information about a meeting, meeting participants and other important items connected to the meeting is managed by Retrace (TechCrunch, 2013e).

“Marusco pointed out, however, that the notes you take are often only useful to yourself, and that email inboxes are a pretty cluttered place for the assignments, reminders, and additional material that is sent afterwards.” (TechCrunch, 2013e)
2.2 The Rational Unified Process

Based on our project analysis and discussions with our supervisor, the Rational Unified Process (RUP) is selected as our software development approach. Kruchten (2003), who was the director of process development at Rational Software (Wikipedia, 2013e), stated that:

“The Rational Unified Process is a software engineering process. It provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of high-quality software that meets the needs of its end users within a predictable schedule and budget.”

Since RUP is an incremental development, there are several significant advantages in RUP. The white paper (Rational Software, 1998) from Rational Software provides six best practices to explain how a system should be developed by using RUP.

**Develop software iteratively**

An iterative approach enables developers to divide a problem into several sub-problems that can be solved iteratively. RUP supports one such approach, which reduces a number of risks in the project (Rational Software, 1998).

**Manage requirements**

RUP manages software requirements by using use case and scenarios. That makes it easier to communicate with customers and provide an efficient way to organise, document and track requirements, as well as trade-offs and decisions (Rational Software, 1998).

**Use component-based architectures**

Component-based architectures are applied to RUP. This approach allows developers to design architectures by developing new components or reusing existing ones. Therefore, software reusability is well supported in RUP (Rational Software, 1998).

**Visually model software**

RUP fully supports Unified Modelling Language (UML) for visualising and modelling software systems in different aspects (Rational Software, 1998).

**Verify software quality**

RUP provides a good means of support for verification of software quality, in order to satisfy the customer’s need and meet the organisation’s standards (Rational Software, 1998).

**Control changes to software**

Since change management is very critical in today’s software development, RUP is capable of accommodating, controlling, and tracking changes in an iterative approach (Rational Software, 1998).

Apart from those significant features in RUP, there are a few drawbacks as well. For example, since RUP is an incremental development approach, it brings difficulties in project management.
“Incrementally developed systems change so quickly that it is not cost-effective to produce lots of system documentation.” stated Ian Sommerville (2011).

Furthermore, Eclipse Process Framework (EPF) provides the Open Unified Process (OpenUP) adopted from the foundation of RUP. EPF (2012) stated “OpenUP is a lean Unified Process that applies iterative and incremental approaches within a structured lifecycle.” Therefore, OpenUP is a very good and improved approach for software engineering projects (Eclipse Process Framework, 2012).

2.3 Software prototyping
In order to demonstrate our theoretical improvement, software prototyping is required in the project. Software prototyping is to discover customer’s needs and search alternative solutions to the problem (Carr and Verner, 1997). The following illustration (Figure 2.1) depicts the process of software prototyping.

![Diagram of software prototyping process]

Figure 2.1: The process of software prototyping

There are several approaches to software prototype development: exploratory approach (throwaway prototyping), experimental approach and evolutionary software development approach (evolutionary prototyping) (Carr and Verner, 1997).

**Exploratory approach**
The exploratory approach is to build temporary models and obtain quick feedback from customers. Throwaway prototyping is the fundamental method in this approach. The purpose of this approach is to identify requirements. The prototype will be discarded providing that the requirement analysis is completed (Carr and Verner, 1997).

**Experimental approach**
This approach is to create prototypes in order to identify the possible partial or full functions in the system (Carr and Verner, 1997).
**Evolutionary software development approach**

Evolutionary software development approach is an effective means of analysing requirements and adapting changes. This approach enables developers to create or improve a system by accommodating changes (Carr and Verner, 1997).

“In contexts of high uncertainty (and unlike contexts with high certainty where waterfall models are employed), the evolutionary prototyping approach dynamically responds to changes in user needs and accommodates subsequent unpredictable changes in requirements, as the development process progresses (Pape and Thoresen, 1992).” stated Carr and Verner.

**2.4 Software evaluation**

For the evaluation of the project, we used evaluation techniques proposed by Lewis and Rieman (1994). This section is to explain the important elements used in the evaluation in detail.

**2.4.1 Participants**

Lewis and Rieman (1994) suggested that participants in an evaluation should be realistic users of the application to avoid misleading result. They also suggested that participants should not be forced to participate in the evaluation. Any sign of hesitation should lead to a cancellation of the process (Lewis and Rieman, 1994, pp.77-78).

**2.4.2 Tasks**

In addition, tasks should be made as easy as possible for participants to perform. Meanwhile, they should not be made too easy since the evaluation tasks may no longer reflect the original tasks and give misleading results (Lewis and Rieman, 1994, p. 79).

“In your test you’ll be giving the test users some things to try to do, and you’ll be keeping track of whether they can do them”. (Lewis and Reiman, 1994, p. 79)

**2.4.3 Test data**

According to Lewis and Rieman (1994), there are two types of test data: bottom-line and process data. Bottom-line data is information of what the test user is doing. Process data is based on observation on participants’ behaviours. They argued that it is more important to know why a test was not successful than that it was not. If the reasons are unknown, it would not help determine where to put the effort of redesign (Lewis and Rieman, 1994, pp.82-84). The method to collect process data suggested by Lewis and Rieman (1994, p.83) is referred to the thinking aloud method.

“You ask your users to perform a test task, but you also ask them to talk to you while they work on it. Ask them to tell you what they are thinking: what they are trying to do, questions that arise as they work, things they read.” (Lewis and Reiman, 1994, p. 83)

It is important to give the test users leading questions such as “Tell me what you are thinking” (Lewis and Rieman, 1994, p.85) to obtain a good flow of comments. Sometimes it is necessary to push the users a little in order to retrieve useful information. This can be accomplished by encouraging the users to keep an ongoing
conversation without giving leading questions. Any assistance given to the users should be recorded and included in the evaluation result. However, the test users should not receive any answers if it is not required for the users to continue the testing. Therefore, to ensure no assistance that may mislead the result, a set of rules should be established before the evaluation (Lewis and Rieman, 1994, pp.82-85).

For some reasons, there could be a value in collecting bottom-line data to compare how long time it takes a user to complete a task or how many errors a user commits. However, it shows that there is a caveat when combining this with the thinking aloud method. Users tend to perform tasks slower when they communicate at the same time, which results in a misleading outcome. Separate tests should be performed to avoid this problem (Lewis and Rieman, 1994, pp.88-91).

Another bottom-line measurement technique is to measure the user preferences by letting the test user give a statement of what he or she thought of the system using a scale of values. The drawback of this is that the results can be based on a variety number of reasons, because different users may rate the application high or low based on different measurements. Moreover, if the reasons why something has to be redesigned are not known, what to change then remains uncertain (Lewis and Rieman, 1994, pp.88-91).

To compare two design alternatives, two common techniques exist. Between-groups experiment tests two different versions of a system on different user groups. Within-groups experiment tests both versions of a system on the same group. In the latter approach, the result may be misleading. It is because the users gain knowledge by solving the problem in the first version which may reduce the time to solve the same problem in the second one. However, if the tasks are simple, there may not be a critical issue (Lewis and Rieman, 1994, pp.92-93).

2.4.4 Summary of test data
To summarise process-data, expected outcome should be compared to the actual result of each user who performs the tasks. Thoughts and comments from the users should be included to examine whether the interface works as expected or not. Data is to be analysed to discover portions of the interface that can be improved. Secondly, errors should be collected along with their impacts (Lewis and Rieman, 1994, pp. 86-88).

Bottom-line data should be summarised as a measurement of time taken from performing the tasks and the number of errors that has occurred. Unfortunately, a set of widespread values can be hard to summarise. To do this, Lewis and Rieman (1994, p.90) suggested a procedure to estimate the range of typical test values. However, it requires a large amount of test users to be able to get a good estimate (Lewis and Rieman, 1994, pp.88-91).
3. Methods
This chapter explains the methods used in the project. The process is described as linear, although it was done by using an iterative and incremental approach similar to OpenUP, mentioned in Section 2.2. Analysis was the focus of attention at the beginning of the project. Design and implementation obtained more attention in the rest of the project. Eventually, the main focus was given to the evaluation. A more detailed description regarding design, implementation and testing can be found in Appendix B.

3.1 Tools
The most important tools used in the project are listed below.

- Microsoft Visual Studio Professional 2012
- Microsoft SQL Server 2012 Express
- Visual SVN
- AnkhSVN
- CamStudio Recorder 2.7
- Microsoft Visio 2010
- Adobe Dreamweaver CS5
- Adobe Photoshop CS5
- Adobe Illustrator CS5

3.2 Setup
For the team to manage the source code and database, a central server was setup with a SQL database and Visual SVN server. AnkhSVN was installed as a plugin in Visual Studio.

3.3 Requirement analysis
The requirements of the application were derived from four different sources. Firstly, the essential ideas were formed by our supervisor. Secondly, those ideas were combined with our own ideas of improvement. A questionnaire was published on the internet to collect statistics of calendar application usage as well as their desideratum. The questionnaire prioritised the requirements as mentioned in the SRS document. The last source of requirements was the requests made by users of Google Calendar on the Google Calendar forum. Full requirements specification can be found in Appendix A.

During the requirement eliciting, mock-ups were created to ease the discussion of requirements with the supervisor by using the evolutionary software development approach defined in Section 2.3. These were used as a base for the implementation. To ease implementation, scenarios were created and evolved in the project which can be found in Appendix A.

Figure 3.1 depicts the relationship between the classes derived from the requirements. Classes marked with green are implemented while the red ones are not. The unimplemented classes are stated as a guideline of how the application is thought to be extended. Each User has a set of Preferences attached to its account. It also has a set of Calendars. Each Calendar contains a set of Events that can be of different types and belong to different calendar services. Currently only support for Google calendar service is implemented. The two different types of events are GLecturerEvent and GClinicEvent which have different event type specific properties defined. Since all events inherit from Event, they also inherit the user specific properties, or Customised Fields, that can be modified by the user and saved to the Preferences. Events can be attached to an external source. For each external source, there exists a class defining the service. ExternalWeatherService makes use of the ExternalServiceUtility to carry
out some of its tasks. Each external service inherits some common properties and behaviour from ExternalService. When updates occur to events attached to external sources, a Notification is shown to the user. Finally, GeventUtility provides CRUD functionality for calendars and events within the Google Calendar service.

Figure 3.1: The logical view (Class Diagram)

Based on the requirements and identified characteristics, the prototype was to be built based on an existing calendar service, i.e. Google calendar. Due to resource and time budget, the prototype had only been considered one or more key aspects contained in the friction.

3.4 Design and architecture
To build the foundation of the application based on the requirements, an architectural notebook was established. The notebook documented patterns, mechanisms, goals, etc. “4+1 architectural view model” created by Philippe Kruchten was applied to the project as a guideline on architectural design. The logical view (Figure 3.1) and development view (Figure 3.2) were the focus of attention. The views were used to form the structure of the implementation along with the stated scenarios. More information about the architecture is provided in Section 5.1.

The following diagram (Figure 3.2) illustrates the components and dependencies in Calendar 2.0 application. The intention of the diagram is to ease the understanding of extending the framework. Hence, it depicts a service-oriented approach in which each component provides a service that can be used by other components. As can be seen, the
main component is the **Framework** component which includes several internal components, i.e., **Entities**, **External Services**, **Preferences**, **Notifications** and **Persistency**. It also connects to some other components such as **External Sources**, **Google Service**, etc. The **Framework** component should be seen as a core component, containing reusable functionality for different type of calendar services. It makes use of **External Sources** through **External Services** such as web services which attaches events to data outside the application. New services can be added in order to deal with data from a specific source. Changes in **External Sources** can notify users through **Notifications**. Different **Notifications** may be created for different purposes. Each user account has a set of **Preferences** attached that are unique for the specific user. The **Preferences** component contains such information. The **Entities** component contains specific event types that can be created to extend the basic events, either using **Event-type or User specific properties**. The **Persistency** component serves as a layer between the **Internal Database** and the components in the **Framework**. The **Google Service** facilitates the usage of the calendar service by providing **CRUD functionality** for events and calendars.

![Component Diagram](image)

**Figure 3.2: The development view (Component Diagram)**

### 3.5 Framework and technology

The framework was built to serve the prototype and to maintain and improve the calendar service itself in the future. Therefore, the framework was created as a set of library classes that can be used by the prototype or future calendar applications. The prototype and framework is separated. The framework is the core of the project solution which consists of several sub-projects.

Based on the requirements and an increasing growth of web application interest, the implementation was carried out as a web application. After a discussion with our supervisor and some research on different tools and platforms, **ASP.NET MVC** project was elected as the main technology of the project.
As the diagram (Figure 3.3) depicts above, the project solution was divided into three sub-projects: Framework, Test and Web. The Framework project served as a self-reliant project and provided basic functionality that can be used by calendar services. The service-specific functionality, which is Google calendar service in this project, was implemented within the Framework. The Test project contained different tests created during the development. The Web project contained the prototype.

3.6 Third-party libraries and mechanisms
In order to carry out the implementation, some mechanisms and other important third-party libraries were considered in the project. In this section, those libraries and mechanisms are described in brief. The detailed information is shown in Appendix B.

- The system must be provided a persistence mechanism in order to store data internally. NHibernate was chosen to provide object relational mapping.
- JavaScript and jQuery were applied on the client side to achieve user interaction.
- The calendar service API was the Google API version 2.
- Forms authentication provided by Microsoft was applied to the prototype. Users were authenticated through AuthSub and given access to Calendar 2.0 applications only if the authentication was successful. Therefore, no user information is stored locally. Controller classes were responsible for verifying requests.
- Testing was carried out using NUnit and Selenium web driver.

3.7 Implementation
The prototype and framework were established in several iterations, each of which fulfilled a set of requirements. The functionality of the prototype closely relates to the one in the framework. Since the framework and prototype were created in two independent projects, it was possible to accelerate the development process. Some more complex processes are illustrated in sequence diagrams found in Section 8.2.6.

3.7.1 Foundation
The project structure was created and the basic functionality of the framework and prototype was to be implemented. In the framework, the domain-specific and service-specific classes were created together with a utility class to manage persistency. The authentication mechanism for Google calendar service was implemented as well. In the prototype, the general structure of the view was created with placeholders for the information to be added.

3.7.2 CRUD operations
Firstly, the functions to retrieve data were carried out, i.e. listing the events and calendars. Those functions were available to the basic types of events, such as all-day
events, and normal events. Secondly, deletion of events was added by clicking the **Delete** button in the form. The functions of creating and updating events were added to the prototype as well. Adding, updating and deleting an event took place in the same form from the top extruder panel, as shown in *figure 3.4*. The form contains basic event information together with information and actions added in Calendar 2.0. The added features are weather forecast service, event type and user specific properties. In addition, a set of test cases were implemented to automate GUI testing, which can be found in the Selenium test project.

![Event form](image)

*Figure 3.4: The event form used for adding, deleting and updating an event*

### 3.7.3 View navigation

Navigation was added using a slider, navigation buttons, and a week view. The slider changes the number of visible days in the view. The navigation buttons changes the start day of the view and the week view changes the view to that specific week.

### 3.7.4 Event type and user specific properties

Event type specific properties were implemented by inheriting from the **GEvent** class, adding more fields to an existing event. They were implemented in such a way that developers can extend them by creating a new subclass of **GEvent**. A more detailed description of this can be found in *Appendix B*.

User specific properties were added as a list to the event. Since **GEvent** inherits from **Event**, all subclasses inherit the feature of extending an event with user specific properties. The properties have attached names which are persisted in user preferences and can be changed by each user under the preferences page. The names can be fetched and saved through the preferences utility.

Since the new fields (from the created subclasses) had no counterpart in the existing calendar application, they were saved in an internal database. The persistency was added in the NHibernate mapping files. Fields that should be persistent was therefore made selectable. The same idea was applied to the construction of the view. Each type of event had its own corresponding view and model. In the controller logic, functionality was added to save and fetch the new event information using the models and views. Event information was added as a part of the event form in the view and shown for each event type correspondingly. CRUD operations were predefined in the framework and could therefore be reused in the prototype. For more information, refer to *Appendix B*. 

- 15 -
3.7.5 Active events
To make the calendar application respond to changes in external sources, it introduced an active support for events. An event was made active by attaching it to an external service. The framework supported this functionality as services by providing an abstract class that new services could inherit from. Both notifications and CRUD operations in cases of changes were made possible to implement.

The implemented service was a weather service. It would provide the user with notifications if the anticipated weather changed based on a location and date. The locations were selectable among a set of cities and countries from which there existed a weather forecast. The notification was made visible to users in order for users to manage the affected event. To make this happen, the method GetMessages returned a set of notifications for all registered services when an event attached to the weather service encountered a mismatch between anticipated and forecasted weather. More information can be found in Appendix B.

3.7.6 Fine tuning
At the end, some shortcomings and minor bugs were fixed and implemented. An example of this is the support for the cross-day events and whole-day events. Convenient functions were added, e.g. add an event by clicking on the timeline, align the view to a specific time, etc. A strategy to manage conflict events was also introduced. The code was commented and packaged in the final submission package.

3.7.7 Testing
Testing was not a major consideration in the project as the final product intended to be a prototype. The tests of the framework were made by using unit testing. Specially, crucial test cases were implemented for the prototype using NUnit and Selenium. More information can be found in Appendix B.

3.8 Evaluation of prototype
The evaluation was performed using the methods described in Section 2.4. Before the evaluations began, the participants were shown a tutorial video to learn the basic features of the Calendar 2.0 application. A Google calendar account was prepared to match the pre-conditions for each task. The participants’ performance on the screen was recorded.

In total, four participants were included in the evaluation. They were between 20 and 25 year old. The nationality, area of study, frequency of usage and gender were divided over all the participants. One participant was from Turkey, one from China and two from Sweden. The gender was equally divided among the participants. Half of the participants use the calendar daily.

The identity of participants remained anonymous. Only necessary information was included in the evaluation result. CamStudio software application was applied to record both screen and microphone. This was to analyse the content of the evaluation afterwards. One of our project’s members filled in a copy of the form based on the records. After all evaluations were carried out, the results were summarised.

The evaluation consisted of five tasks, four of which were to be done in both Google Calendar and Calendar 2.0 application. The last task was presented to test users. For each task described in Appendix C, there is a following scenario for the participants to make the task more realistic. The participants explained their opinions when performing the tasks. In addition, there is a set of questions for each task. At the end of the evaluation, participants were given four general questions about the prototype.

Furthermore, each task has expected outcome and reasons for testing. The pre-conditions have to be fulfilled before testing. The expected paths (how users are
expected to perform different tasks) were described in the expected outcome. The evaluation results can be found in the following section.
4. Evaluation results
This section summarises the result from the evaluation by combining the results from each user for each task. For the first three tasks, a navigation path diagram depicts the paths chosen by users in both Calendar 2.0 and Google Calendar. Each task also contains an aggregated answer. Finally, the section summarises the answers regarding the general questions. The team member performing the user test is referred to as the observer.

The questions and task explanations are replicated in the result. More information regarding the content of the evaluation form is shown in Section 8.3.

4.1 Task 1: Week navigation
The first task involves week navigation by letting users navigate to a specific week in the view. The characteristic encountered in the task is first and foremost the dynamic view.

Scenario
A customer calls and wants to book a meeting at week <week no.> between 8 am and 5 pm between Monday and Friday.

Task
Your task is to find a suitable time for that meeting to take place.

Figure 4.1: Navigation paths for task 1

All users argued that Calendar 2.0’s week numbers were more noticeable than Google Calendar’s. Some users argued that the lack of noticeable week numbers in Google calendar forced them to use the monthly view for the task instead of the week view. However, all users thought the Calendar 2.0 made the task easier to perform due to the week navigation view.

Most users, especially the Swedes, found the week navigation view useful in order to be aware of what happens in a specific week. They also found it useful to be able to navigate to a specific week in the view since they organise their time week wise.

Figure 4.1 depicts how users navigated in Google calendar and Calendar 2.0. It shows that most users used the next week arrow to get to the specific week in Google calendar. In Calendar 2.0, users mostly used the week view in order to navigate to the corresponding week.

Users suggested two improvements. Firstly, the navigation buttons could be improved to make the navigation easier and to make the navigation more distinct in
relation to weeks. Secondly, a month view could be added to improve the navigation options even further.

4.2 Task 2: One week planning
In task 2, users are asked to make plans for the upcoming week by finding suitable times for a meeting with a friend. The characteristic considered is first and foremost the dynamic view.

Scenario
A friend asks you to have a beer with him. He arrives tomorrow and will then be in town for a week. Since he is working during the days, only evenings are feasible meeting times.

Task
Your task is to find a set of suitable times when you can meet with him while he is in town.

Figure 4.2: Navigation paths for task 2

In general, users preferred Calendar 2.0 since they could use the slider to see all days needed to perform the task in one view only. However, some users argued that Google had an advantage in the alignment of the view, which avoids scrolling. Most users also thought the navigation bar and the dates should follow the scrolling of the view.

Figure 4.2 depicts the navigation paths for Google calendar and Calendar 2.0. In Google calendar users tended to navigate to the month view or get stuck in the week view. However, this is not directly seen in the diagram. In Calendar 2.0, users used different approaches to carry out the task. In Calendar 2.0, users did not require any feedback from the observer to carry out the task.

Users suggested that the time could be divided into time slots that depicts free and busy times using different colouring. Time slots could also have different colouring for different purposes.

4.3 Task 3: More advanced, two week planning
Task 3 is an extension of task 2 in which users are asked to carry out a more extensive planning task over two weeks in the calendar. The main characteristic is the dynamic view.
**Scenario**
Your boss asks you to carry out a task that will take you about 40 hours. You work between 8 am and 5 pm. Assume work days are Monday to Sunday.

**Task**
Your task is to find out whether this can be performed within a 2 weeks range or not.

![Diagram of Google Calendar](image)

1 participant

![Diagram of Calendar 2.0](image)

4 participants

Figure 4.3: Navigation paths for task 3

Users preferred Calendar 2.0 since they thought it was easier to carry out the task using a 2 week view instead of changing views. Some users also argued that a lot of unnecessary information exists in Google calendar’s view which makes the task harder to accomplish.

All users used the same path in Calendar 2.0, as seen in Figure 4.3. In Google calendar on the other hand, navigations are distributed over many different views and actions. Users struggled with performing the task in Google calendar and input from the observer was required for most users to perform the task. However, in Calendar 2.0 users did not need any input from the observer and did manage the task easily.

For this question, answers were scattered. Some users thought 2 weeks is enough in the view and argued that it is easy to change views using the navigation buttons. They stated that normally there is no need for a larger view. The same users claimed that the view should be too compressed and that the information should be hard to distinguish.

Other users suggested that more days could be added to the view adding a feature that increases certain parts of the calendar to avoid the problem above. The month view, using the same consistent view, could be useful when planning diets, etc. according to one user.

4.4 Task 4: User specified properties
Task 4 lets users find notes attached to a specific lecture to prove that shortcomings exist in the Google calendar. The main characteristic involved is **personal**.

**Scenario**
You subscribe to a calendar with lectures for a course you are taking. Two weeks ago, <Day> <Date> you made some notes that are related to that lecture. The lectures name is <Name>
Task
Your task is to find the notes related to that lecture.

The results show that no user knew of this limitation before. They normally saved information separately without attaching them to events. After users realised that Calendar 2.0 had this feature, all but one agreed that this would be a useful supplement. The same user argued that no more than simple paragraphs were to be added as information on events. Additionally, the rest of the users required the ability to add attachments to such events as well.

4.5 Task 5: Active Event
This task covers the active event part of the characteristics. Users are shown how an event can attach a weather service and let users be notified of the changes. The characteristic in friction that is covered is active.

Scenario
You are going on a picnic on <Day>. Of course, you want it to be sunny. If the weather changes to something else, you want to be notified so that you can change your plans.

Task
Create an event that notifies you when the forecasted weather changes from sun (is something else than sunny). When that happens, you should modify the event, i.e. change date, or delete it. This task is done by the observer.

In general, users thought the weather forecast was a good supplement to the calendar and something that they would use if it existed. Additionally some users argued that the notifications should be shown when the event is about to happen. Some also argued that the representation of the notification stack could be changed and that the notification should disappear after deleting or changing the event. Other suggestions, such as marking events that are affected by the external services or related to notifications, were proposed by one user.

Users had some different suggestions of features to add regarding external services:
- Events related to travelling, such as trains, flights, bus, etc. could be attached to external services.
- Relate events to react on line length change, such that the line to a store gets too long. The user suggested that the event could be attached to the store in order to react if the line gets too long.
- Booking medical appointments, hotels, flights, etc. within the event form to be attached with the external booking system.

4.6 Answers to general questions
Calendar 2.0 was the choice of all users with the motivation of its consistent view and easy navigation. Users also thought it was easier to use while planning. One user thought that static views could have an advantage in some specific cases.

All users agreed that the conflict management, as depicted in Figure 4.4, in Calendar 2.0 was more appropriate than the one in Google because of the amount of text that can be shown using this approach.
Users did favour Calendar 2.0 due to features such as week navigation and weather forecast service. They also argued that the user interface is more user-friendly than Google’s.

Some ideas from users that can be added to Calendar 2.0:

- Focusing the view on current time to emphasise events that is more nearby in time. Avoid showing uninteresting events in the view.
- Plan meals and produce shopping lists to make day-to-day work easier when the matter comes to daily workloads.
- Make a better visualisation of information in the calendar, e.g. creating a first page with a summary of events, free and busy information, etc.
- Pictures could be tagged and added to the timeline.
- Add other external services to the calendar.
5. Discussion
The discussion section intends to discuss the result of the project to the research question and to position the prototype and findings in the scope of calendar-related applications. At the end of this section, the limitations of the findings as well as future work are stated.

5.1 Evaluation result and research question
As mentioned in Section 1.2, calendar applications have the deficiency of the characteristics: dynamic view, intelligent, personal, active, information-centred and extensible. In this section, we discuss the result of the evaluation and direct it to the problem statement. Each task was created with the characteristics in mind as they have been stated in each task separately. Some have received more attention in the project than others. We do not replicate the answers from the users, merely making an argument of the reasons why we think characteristics were mitigated.

The most important characteristic considered in the project is the dynamic view. Task 1, 2 and 3 evaluates this in different aspects. Feedback from the users regarding the view was predominantly positive. This can be seen in the navigation paths and users’ reaction when carried out the tasks. Hence, we claim that Calendar 2.0 has an advantage of Google calendar when performing these kinds of tasks, thereby mitigating the friction in the aspect of dynamic view by providing a consistent view.

Task 4 mainly focuses on the personal characteristic of calendar application. Only a small part of the evaluation is considered for this aspect. Calendar 2.0 has more features that can make the calendar more personal, such as event type specific properties and deletion of read-only events. Results from the evaluation have a positive outcome. Users admit to use the feature if it existed in a calendar application. We therefore state that it would reduce the friction in the personal aspect if such feature existed.

Task 5 covers the active part, in which users were shown the weather forecast service. Users agreed that it was a good supplement to existing calendar applications. The value of calendar applications increased according to users, even if some shortcomings and improvements were suggested. Moreover, we also received several proposals of other services to integrate. This shows that users expected more of calendar application in this area. Hence, the active part is an aspect, if paid more attention to, which could mitigate calendar friction and increase the value of the application.

Intelligent and Information-centred was not considered explicitly, however they are involved implicitly in some evaluation tasks as well as in the prototype.

Extensible relates to the reusable components provided in the framework of Calendar 2.0. No evaluation was carried out for this characteristic, although a number of functions rely on it. In addition, it gives future developers a great value.

Why does the friction exist? Our hypothesis in Section 1.2 states the friction is the deficiency of some characteristics of calendar application. One obvious reason is the slow development of calendar applications, which less improvements have made in calendar applications. This causes friction of usage. Our evaluation result shows that many users are used to the limitations of calendar applications and are positively surprised when improvements are made. If users respond positively on the changes, the questions are: why has the friction not yet been mitigated? Could it be that users are accustomed with this functionality so that major changes could increase the friction? How much interest must the users show in order to encourage the software giants in such development?

Can it be mitigated? We claim that it can be. The evaluation result yields a positive outcome. The evaluation tasks are related to characteristics of calendar application...
which in turn are related to the implemented functionality. However, we do realise the limitations of the project. This is far from a complete study of the field and does not involve a mitigation strategy for all derived characteristics.

Figure 5.1: A mitigation strategy

*Figure 5.1* depicts the strategy of friction mitigation used in the project. We started out with defining friction as a set of characteristics. This resulted in some requirements which were realised in an implementation. Based on the implementation, we carried out an evaluation. The evaluation was related to the characteristics defined at the beginning of the project by its outcome.

**If the answer is positive, what impact will that have on the usage?** Based on earlier discussions of the evaluation result, we claim that the impact is positive. We argue that there will be a huge positive impact if more attention is paid to calendar application in order to mitigate the existing friction, as seen in the evaluation result in *Section 4*. This will help users work more efficient in their day-to-day work. However, the magnitude of the impact cannot be unambiguously determined from the findings in this research.

5.2 Discussion of the chosen methods

In this section, we briefly discuss why we chose certain methods in the project rather than others. Each paragraph covers one subject exclusively. Only methods having a considerable impact on the project are discussed.

Firstly, we would like to discuss the choice of selecting Google calendar API version 2 ahead of version 3. At the beginning of the project the version 3 of Google calendar API were still in an early stage. Not much documentation or example code existed. We discovered this could be a huge risk in the project which was mitigated by using version 2. However, to adapt the framework functionality to version 3 should not be too much of work, since many implementation details are alike and simpler. We have studied that, since we did have a test implementation working for desktop applications. Some operations exist in the binaries attached to the project. However, we do not leave any guarantee of their stability, since they have not been used in the project.

Google calendar was chosen from its competitors due to several reasons. Mainly Google provides an API developer’s guide and a forum contains comprehensive instruction. It is an online service, which makes it easier to integrate. There is no additional installation required on a client machine. At the beginning of the project, our evaluation method was not yet decided. Since our university is currently using Google as a supplier of the calendar service, this would benefit our evaluation. In addition, we also realised some advantages that we can gain by learning more about Google calendar API since we use the calendar daily. From the first day of the project, Google calendar has been the centre of attention, which made it the natural choice.

As to database persistency technology, we chose between Entity Framework and NHibernate in the project due to several reasons. We finally selected NHibernate due to three reasons mainly. Firstly, we thought NHibernate provided a more dynamic mapping, since we were to have persistent object in different sub-projects. Making this work with Entity Framework could be harder. Secondly, NHibernate dynamically drops
and creates the database at run time which is a huge advantage when both team members simultaneously develop the project at different locations. Thirdly, bad experience in Entity Framework from earlier projects made us reluctant to Entity Framework.

We chose to create a web application due to the simplicity of making interactive user interfaces, as the number of web applications increasingly grows in the software market. Because of the increasing popularity of .NET, Visual Studio and the MVC pattern, we felt the need to expand our knowledge in the field of software technology. .NET framework includes most of the functionality needed in order to carry out the implementation. Other functionality can easily be added using NuGet. Other programming languages may also suffice, such as Java, although the support for building applications using the MVC pattern is not as sophisticated as in ASP.NET. Another cause that makes a huge difference in the speed of and quality of development is the choice of IDE. Visual Studio gives a better support, in our opinion, than Eclipse, mostly due to the integrated functionality of managing models, views, controllers and their relationships. Visual Studio also gives a great support in GUI design. Third party solutions normally exist for other IDEs to make up for the shortcomings, even though those normally make the project more difficult to manage because not all tools are gathered in the same application. One such tool is Play Framework for Java. The choice of C# thereafter fell naturally since both team members have good knowledge of the language which is widely used in the software market.

The purpose of choosing MVC pattern includes the following three main concerns. Firstly, it is much easier to manage the complexity of the system by using different layers (Microsoft ASP.NET Team, 2013). Secondly, the separation of layers enables developers and web designers to work independently. That, moreover, provides flexible testing in different layers (Microsoft ASP.NET Team, 2013). Thirdly, changing user interface does not affect the model layer (MSDN, 2013). That means the user interface of the system can be modified without changing the business logic. Therefore, using MVC pattern brings a great maintainability to the system.

5.3 Project positioning
In this section, we position the project findings in the bigger scope of calendar and calendar related applications. In previous work, we relate the project findings to existing calendar applications. The rest of the section positions Calendar 2.0 in the scope of calendar related applications and briefly describes its general limitations.

5.3.1 Previous work
The project focus was to mitigate friction from existing calendar applications. Previous work therefore relates to current calendar applications. The three calendars under consideration was Yahoo, Google, and Outlook calendar. All of them include the notion as we refer to calendar friction. Recall that friction was the deficiency of some characteristics of the calendar application. We consider the friction as something more than just a lack of features. It is an obstacle causing calendar applications diminish the development of other calendar related applications.

Functionality in Calendar 2.0 is implemented based on the functionality in Google calendar. Thus, Google calendar as an existing calendar application is compared to Calendar 2.0 in the evaluation. Calendar 2.0 should not be considered as a replacement of Google calendar, rather a new layer added on top of Google calendar. That suggests that Calendar 2.0 enhances some functions from Google calendar and introduces some new functions. Hence, the positioning of Calendar 2.0 should be in relation to Google calendar in general. If the existing calendar applications are considered as calendar 1.0,
our project Calendar 2.0 on a larger scale (Figure 5.2) can be treated as a new layer upon other calendar services as our framework intends to be.

![Calendar Desktop/Web/Mobile Application](image)

**Figure 5.2: The layer of calendar application**

### 5.3.2 Virtual personal assistants

As mention previously in Section 2.1, virtual personal assistants and calendar applications share common features. Hence, it is necessary to discuss their differences and similarities. To a large extent, problems that appear in the existing calendar applications can be solved by some personal assistant applications. We conform to this statement since personal assistants are getting sophisticated and intelligent. As an example, Sherpa provides users with weather service for a specific day as a result of user requests. This is very similar to the active weather service in Calendar 2.0 application, although more convenient in many cases.

A caveat for virtual personal assistants that exists today is the lack of knowledge of what users expect from the application (TechCrunch, 2013b). This becomes even more obvious by observing the widespread categories of features in applications such as Sherpa (Google Play, 2013c).

We argue that, by making use of the existing calendar applications, the transition can be made much simpler. This can incrementally evolve an existing calendar application into a personal assistant. TechCrunch (2013b) states “There’s also the issue of finding an app that fits everyone’s lifestyle”, which amplifies this argument.

We also argue that the calendar’s purpose is to organise a user’s arrangements when considering time and place. Thus, this type of information should be located in the calendar. Calendar applications and personal assistant can exist in collaboration with each other whereas calendar applications serve as the base of information and virtual personal assistants the user friendly interface. Hence, findings in this project can be used to improve previous calendar applications and thereby also virtual personal assistants.

### 5.3.3 Meeting notes applications

As mentioned in Section 2.1.3, an application named Retrace brings the notion of meeting notes to a new level. The founders stated that the app helps you prepare for meetings and get there on time, but no app focus on the follow-ups (TechCrunch, 2013c). Another interesting aspect of Retrace is the integration with Google calendar events. Retrace recognises the advantage of adding an extra layer on top of the existing calendar applications.

Calendar 2.0 application implements the notion of event type specific properties and user specific properties by adding a new layer on top of an existing event in Google
calendar. This makes the information in the calendar application reusable. We observe that keeping a familiar user interface makes it easier for people to use, which has been acknowledge in the evaluation result in Section 4. Moreover, information is kept in a central storage to be used by many integrating services.

As to the features concerning meeting organiser, Calendar 2.0 application cannot be compared to a sophisticated app (e.g. Retrace). However, it is still clear that existing calendar applications have insufficient features to keep track of event information in calendars. Therefore, a new application is required in order to execute such tasks. We suggest that this feature should be added as a plugin to the calendar instead of a separate application. Although we do not state that Calendar 2.0 should replace any of the existing applications, solely propose the possibility of adding the same features inside the calendar applications instead. The findings in this project indicate that this could increase user’s value.

5.3.4 Limitations of findings

Even if some limitations were made in Section 1.4, the scope of the findings might be somewhat more limited due to the huge scope of calendar applications and the restrictions made in the process. To be able to carry out a full research, more time, and resources are needed.

The characteristics of friction used in the project, as well as the one paid more attention to, might unintentionally limit the outcome. To avoid this, we suggest a more comprehensive requirement elicitation to be carried out. Another limitation is the number of calendar applications considered in the project. The findings are limited to the Google calendar service since those were the compared calendar in the evaluation. A more extensive research, including more calendar applications may have given another outcome. The types of calendar related applications considered as well as the chosen platform might affect the outcome of the project.

User testing was carried out on a limited amount of users; hence the results could be limited to the type of users. To be more accurate a larger group of users is needed. However, in general the findings are enough to draw the conclusions needed to derive useful information from the project.

5.4 Future work

As described in Section 8, some requirements were not implemented in the project. To continue the work of this project, more attention could be paid to those in general. Specifically we suggest the reader to put more effort into the TimeLine since we argue that this will make the calendar the central storage of information. Another interesting feature is the talking reminder which is closely related to features seen in virtual personal assistants.

The prototype were created as a web application in .NET, however it might be of interest to adapt it to Android or IOS since apps are popular these days. It might also be of interest to port the project into another programming language, specifically the framework. And even making it open source.

External services introduced in the project have a huge amount of variations. We introduced the weather forecast service, yet several other examples exist. As an example, we considered integrating SJ’s timetable (a Swedish train company) and flight information. By extending the support for subscriptions to external services and adding functionality for developers to create plugins, the calendar application could evolve rapidly.

In the project, we also introduced user specific and event specific properties. Current calendar applications do not support personal views, instead each event are treated as the same. We argue that this is not the case in real life. By extending and structure the
information of events, the calendar could not only be made more central but also easier to use.

Planning is a central concept that can be of interest. Many tasks used in the prototype evaluation included planning to some extent. The difference between Google calendar and Calendar 2.0 was clear in this area. Hence, there is a need to evolve the current tool for planning. Some ideas are made in the requirements in Appendix A and also refer to the evaluation results in Section 4.

Great thought has been put on the view to remove the static views of current calendar applications. We suggest that even more attention could be put on the view in order to make it even more efficient.

Security was not one of the high priority requirements in the project since the end product was to be a prototype. However, adding a second layer of authentication in which users have to authenticate to the internal system as well, will increase the security even further. Although the current authentication procedure is identical to the one used by Google themselves. The communication with Google calendar service could be encrypted as well, in order to protect the information sent and received.

In addition to the requirements gathered, more can be added to this area, i.e. to mitigate the friction. As we came to notice while carrying out the project, calendars and virtual personal assistants are all the rage at the moment. The following statement from TechCrunch (2013b) emphasises the interest of investigating the area further: “As Drew says, all the technology is there — and yet, no one has really pulled it all together in a way that makes these apps truly smart”. We argue that, by adding features to the current calendar applications, specifically the most used ones, a lot of progress can be made to accomplish the dream of a frictionless calendar.
6. Summary and Conclusion
The friction was mitigated by reducing the problems caused by the deficiency of the
characteristics defined in Section 1.2. In order to address these shortcomings,
requirements were elicited, and a prototype and framework were created. The dynamic
view, personal and active characteristics were evaluated by carrying out a set of tasks.
A positive feedback from participants showed that the prototype mitigated friction
associated with these characteristics. The characteristics Information-centred and
intelligent were not considered in the prototype and evaluation. Moreover, extensible
characteristic was not evaluated but implemented in the framework as reusable
components. Hence, we concluded that calendar application friction can be mitigated by
addressing the deficiency of the characteristics.
References


Appendix A: Requirements
The requirements stated in this section are the basic requirements for the prototype and framework. The requirements are divided into different categories, each of which concerns different aspects of friction. Each category is prioritised as high, low or medium. Items in each category are further divided into three types as shown below. Items marked with red or orange colour can be considered in future work.

- Green items are fully implemented.
- Orange items are partially implemented.
- Red items were omitted.

Note that the scenarios are not derived from user requirements. Instead, they are used as a support for the implementation.

Since both requirements and scenarios evolve during the development process, the requirements and scenarios is stated in their most recent versions.

Requirement list

Events associated with external services

<table>
<thead>
<tr>
<th>Priority</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Active, Personal, Intelligent, Information-Centred</td>
</tr>
</tbody>
</table>

- The system must let users choose external services to subscribe to.
- The system must let users modify the subscription of events attached to external services.
- The system could notify users of the change of events attached to external services.
- The system must provide different services as plugins.
- The system must be able to manage events from external sources.
- The system could let users choose what to do when events from external sources are changed.
- The system could perform operations on events from external sources in order to add additional information to the calendar.
- The system could make sure that the time span for events from external sources is in the required time span for the type of event.

Events with user specified and event-type specific properties

<table>
<thead>
<tr>
<th>Priority</th>
<th>Focus</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>Personal, Information-Centred</td>
</tr>
</tbody>
</table>

- The system must provide a way for users to specify user specified properties for an event.
- The system must provide functionality for users to save user specified property names for later use.
- The system must provide different types of events, e.g. Lecturer with different event type specific properties.
- The system should let users add information to read-only events, e.g. events in a subscribed calendar.
Customised reminders

<table>
<thead>
<tr>
<th>Priority</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Personal</td>
</tr>
</tbody>
</table>

✗ The system must provide alternative type of basic reminders, including: sending email, opening documents.
✗ The system must provide an advanced type of reminders such as starting applications, speak event information, etc.

Views

<table>
<thead>
<tr>
<th>Priority</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Dynamic View, Personal</td>
</tr>
</tbody>
</table>

- The system must provide consistent views regardless of time spans.
- The system must provide information that indicates the current day and focuses on the current day.
- The system must expand the view according to available space.
- The system must provide users with the ability to change views by increasing and decreasing days and a set of days.
✗ The system could use colours to depict the “busy” status for each day.
✗ The system could depict different time spans of the days.
✗ The system must provide a detailed day view, containing all events and objects and their relations in which the visibility for different types of objects can be turned on or off.
✗ The system must provide an event view in which the event information and its attachments are shown.
✗ The calendar could include a scrollable text containing events that are to occur at a specific day.

Timeline

<table>
<thead>
<tr>
<th>Priority</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Information-Centred</td>
</tr>
</tbody>
</table>

✗ The system must provide users with a timeline for objects with their time related properties.
✗ The system must let users add objects to the timeline.
✗ The system must provide an adequate view for objects on the timeline and its relations.
✗ The system could let users manage time-related properties for objects attached to the timeline.
✗ The system could provide functionality to automatically update object information on the timeline from a source of information.

Settings

<table>
<thead>
<tr>
<th>Priority</th>
<th>Medium</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Personal</td>
</tr>
</tbody>
</table>

- The system could have a list of settings indicating the action to take when an external resource causes an update to an event.
✗ The system must let users change the time spans of days.
The system could have a list of settings whether planned tasks have to be verified or not.

**Planning assistance**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focus</td>
<td>Intelligent, Personal</td>
</tr>
</tbody>
</table>

The system must be able to create a proposed plan for users based on events and tasks from different sources.

The system must let users prioritise different sources.

The system must let users change the proposal before saving.

The system must help users to solve conflicts.

The system must plan events within the specified time spans.

The system must organise events based on time, importance, prerequisites, etc.

The system must let users choose which categories of events and tasks to include in the planning.

The system could let users add other requirements such as breaks between events.

The system could let users change the plan after applying it.

The system could let users include other predefined events such as dinners, sleep.

The system could allow all-day tasks and apply them as a task list.

The system could allow users to set tasks as a part of an event instead of individual tasks.

**Scenarios**

**Create event**

Users create an event by either clicking the event form button or by clicking on the timeline. They can then insert information for the event that belongs to the event. Users can at any time insert information in all fields, including event-type specific properties and user specified properties. However if the event is read-only, only event-type specific properties and user specified properties can be inserted. The event may also be attached to an external service, making it active.

**Delete event**

To delete an event, users click on the event in the view and click the delete button. They are asked whether they want to delete it or not. If accepted, the event is deleted. In case of a read-only event, the event is only marked as deleted internally.

**Modify event**

To modify an event, users click the event in the view. They then change the information on the event in the same manner as when creating a new event.

**User specified property names**

Users specifies the property names for the user specified properties on the settings page by inserting its corresponding value and press the save button.
**Toggling calendars**
To toggle calendar visibility users click a calendar in the left menu. The colouring is changed and the calendar is made visible or not depending on the previous visibility.

**View navigation**
The user navigates in the view by using the arrows, week view and slider. The “<” and “>” arrows lets the user navigate one day at the time. The “<<” and “>>” lets users navigate one view at each direction. The slider let users change the number of days in the view. The week view let users navigate to a specific week in the view.

**Notifications of changes of events attached to external services**
Notifications are shown to users when changes occur in an external source for events attached to external services. A notification is created which let users be aware of what happened. The user can click on the notification to modify the event. In the prototype the weather service is implemented as an example of external service.

**Login**
Login is taken care of by redirecting the user to the Google login page. After authentication, the user is redirected to the Calendar 2.0 application.

**Logout**
The user logs out and is redirected to the first page.
Appendix B: Guidelines for future development

The intention of this document is to guide future developers of the Calendar 2.0 application. It contains information about the framework and how the prototype realises the framework. Consider this is a development guideline rather than a full documentation of the application.

Class diagram

![Calendar 2.0 Application [Class Diagram]](image)

Figure B.1: Calendar 2.0 class diagram
Component diagram

**Figure B.2: Calendar 2.0 component diagram for the framework**

**Framework**

**Project structure**
The solution is divided into several projects. This mainly separates the functionality of the framework and prototype. The framework (*Figure 8.3*) is further divided into one part that is the framework’s base and the other one which tightly collaborates with Google’s calendar service. To add support for a new calendar service, developers simply create a new project making use of the functionality of the base framework and Google service framework.

**Figure B.3: The structure of the framework project**
The base framework contains basic functionality for Calendar 2.0. It contains reusable components such as External Services and Persistency as depicted in Figure 8.2. Calendar service related functionality is contained in the Google Service component.

**Event-type specific properties**
Each event may retain event-type specific properties, which may be excluded from the underlying calendar service. To address this requirement, Calendar 2.0 lets developers create subclasses of the service-specific event class, which in turn inherits fields from the base event class in the framework. The operations provided by the Google Service component enable developers to manage both internal and external data without caring for where the information is persisted. To make use of the functionality, a new subclass of GEvent should be created and its fields should be added in the file “Events.hbm.xml” for mapping.

The prototype implements this feature by creating two specific events; GLecturerEvent and GClinicEvent. Both inherit from GEvent which in turn inherits from Event class. The event type can be changed in the event form at any time. As the prototype is implemented using the MVC project in visual studio, each type of event has its own view and model. The form data is fetched and posted using jQuery.

**User specified properties and user specified property names**
User specified properties are a part of the Event class. If an Event object has values for user specified properties, they are persisted automatically in the database. Since they are implemented using a List, properties can be attached to an event without numerical restrictions. They differ from event-type specific properties in that users can manually set the names for user specified properties in preferences.

User specified properties and names can be set using the preferences utility. Each user has attached at most one Preferences object (none if not created). In Preferences class, there is a set of CustFieldsType applied to each user. Different event types have different CustFieldsType. Hence, user specified property names are applied for each user and event type.

Properties and names are kept in two different locations. The properties are a part of the Event object and the names part of the user’s settings. This requires developers to be careful when using this feature since the lists has to be synchronised manually. However, utilities facilitate the usage remarkably. Figure 8.4 indicates the usage of the PreferenceUtility.

```
Preferences preferences = PreferenceUtility.GetPreferences(GServiceUtility.GetUserFromService(service));
```

Figure B.4: An example of PreferenceUtility usage in Controller

**External services**
To help developers add a new external service, there is an abstract class that defines a set of fields and methods. GetMessages and Refresh methods are fundamental. GetMessages method is to return a set of notifications which the service has created for a user. Refresh method is to retrieve current data for external services. As to how this should be done, it depends on the service and which type of event information executes the task. It may just update the notification status for GetMessages method to return a valid set of notifications. Alternatively, it may perform another action, which can be anything from events management in a calendar to email sending.
NHibernate configuration
Calendar 2.0 application facilitates the database integration by the use of NHibernate. The base settings file “hibernate.cfg.xml” is located in the framework project. It contains the database settings. Mapping files for persisted classes are contained in their corresponding project. For a new project, a new mapping file has to be created. If the class is a subclass of an already existing class (which is almost always the case in this project), only the fields in the new class have to be specified. Figure 8.5 displays an example of usage.

```xml
<subclass
    name="Calendar2.GoogleService.SpecificEntities.GClinicEvent"
    discriminator-value="Calendar2.GoogleService.SpecificEntities.GClinicEvent">
    <property name="PatientName"/>
    <property name="PrepNotes"/>
</subclass>
```

Figure B.5: An example of Hibernate subclass usage in Events.hbm.xml

The **NHibernateSession** class manages to initialise the session by taking a set of assemblies which the mapping files adhere to. The **initialise** method is invoked using the assemblies in the mapping files. After initialisation, the **GetSession** method returns the active Hibernate session. Figure 8.6 displays an example of usage.

```csharp
using (ISession session = NHibernateSession.GetSession())
{
    using (session.BeginTransaction())
    {
        ev.EventId = newEvent.EventId;
        session.SaveOrUpdate(ev);
        session.Transaction.Commit();
    }
}
```

Figure B.6: An example of Hibernate session usage in **GEventUtility**

Before invoking methods in the utilities that are connected to internal data, make sure that the Hibernate session is initialised by calling **NHibernateSession.Initialize (assemblies)** where the parameters **assemblies** specify the assemblies of the mapping files to include. If the assemblies are not in the mapping files, an exception will be thrown.

Google Service
Currently, the application is solely integrated with the Google calendar service. It includes entities and utilities to support basic calendar functionality. The **GCalendarUtility** class contains two methods: **GetList** which returns a list of calendars for the logged in users and **ToggleCalendar** which determines calendars’ visibility. The **GEventUtility** class contains several methods which can be summarised as the CRUD operations for Google events. The **GServiceUtility** class is rather trivial and only contains a method that returns the current user from a specific service object. This is used to keep track of the currently logged in user. No further authentication is needed, since the Google calendar account is used to authenticate the user through the **AuthSub** authentication system.
Preferences
The framework contains a **Preferences** class that holds the preferences for a specific user. The **PreferenceUtility** class contains methods to fetch and save the preferences to the database (*Figure 8.4*), i.e. to make them persistent. At the moment the preferences contains the user specified property names. Preferences information should be added to this class or to related classes. To make them persistent, the fields should be added to the mapping file “Preferences.hbm.xml”.

Prototype

Project structure
MVC (Model-View-Controller) is applied to the prototype project (*Figure 8.7*). The view part of MVC consists of the JavaScript files, master page and view pages. The master page contains the main structure of the web application by using placeholders for information to be fetched partially. Most of the portions are populated using Ajax, whereas the server renders HTML that the client appends to a specific div. By doing this, refreshing the whole page is not completely necessary. Further information can be found in the section View and jQuery/JavaScript.

![Prototype](image)

*Figure B.7: The structure of the prototype project*

The controller part consists of several specific controllers:
- **LoginController** – handles log-in and log-out operations for the users.
- **ViewController** – manages log-in and log-out operations for the users.
- **PreferencesController** – manages to fetch and save preference data.
- **NotificationController** – manages notifications displayed at the bottom-left corner.

View
The view is divided into different components that perform different tasks. Although the components are separated, some are dependent on each other. A further discussion about the components and their implementations is carried out below.
The day navigation is done by using the “>”, “<”, “>>”, “<<” or “Today” buttons at the top of the web page. Hot keys are also available. More information is provided in the user manual and the file HotKeys.js. In practice, all navigation buttons change the start day of the view by modifying the day0 variable.

The week navigation is done by clicking the week numbers. When a week is selected, day0 is set to be the first day of that week. The first day of the week is calculated by starting to count from the last Monday of the last week of that year. To add more years into the week view, add a new Header in the accordion. Figure B.8 displays how the week view of 2013 is created.

```html
<ajaxToolkit:AccordionPane ID="Pane2013" runat="server">
  <header>2013</header>
  <content>
    <div id="scroll2013" class="weekScrollViewer" data-monday="23" data-week="52" data-year="2013">
      for (int i = 52; i > 0; i--)
      {
        %
        <a class="selectedWeek" data-week="<%= i %">Week <%= i %>\</a>%
      }
    </div>
  </content>
</ajaxToolkit:AccordionPane>
```

Figure B.8: An example of week view for 2013 in Index.aspx

Events in the view are rendered based on the number of days in the view, numDays, and the value of day0. As mentioned above, day0 is set by using the navigation buttons and numDays are set by using the slider.

**jQuery/JavaScript**

The client side logic is mainly carried out using jQuery and plain JavaScript. This is to increase the interactivity and to avoid page refresh. The script files are restricted to one portion of the application as far as possible.

- Event/Form – manage all form operations such as posting, fetching and making the form elements interactive.
- Event/Hot Keys – activates and deactivated the hot keys that can be used in the calendar view. Hot keys are currently unbound while the event form is active.
- Event/Navigation – manage the navigation, week view, slider, etc.
- Event/View – manage events’ rendering and view interactivity.
- Login – functions concerning the login page.
- Preferences – functions concerning preferences.
- Site – general functions used in different parts of the application.

**Authentication**

Authentication is carried out through Google authentication service AuthSub. A user logs in by using his or her Google account. Google redirects the user to a specific URL set up by the prototype, which retrieves a token and creates a valid service that can be used for subsequent calls. The framework can also be used together with the ClientLogin by creating a new Service object and applied the user credentials. This approach is used when testing the framework.
public ActionResult LogIn()
{
        "https://www.google.com/calendar/feeds/",
        false, true);
    return Redirect(redirectUrl);
}

public ActionResult ReceiveAuthentication(string token)
{
    IAuthSubRequestFactory authFactory = new IAuthSubRequestFactory("e1", "CalendarSampleApp");
    authFactory.Token = AuthSubUrl1.exchangeForSessionToken(token, null);
    ICalendarService service = new ICalendarService(authFactory.ApplicationName);
    service.RequestFactory = authFactory;
    Session.Add("calendarService", service);
    Session.Add("sessionToken", authFactory.Token);
    FormsAuthentication.RedirectFromLoginPage("someName", false);
    return RedirectToLocal("/");
}

Figure B.9: An example of authentication in LoginController

Events
Events are divided into four categories: all-day event, whole-day event, cross-days event and normal event. The difference is explained in the glossary. All-day events are represented on the top of the calendar view. Whole-day events are represented in the view starting at 00:00 and ending at 23:59 on the same day. Cross-days events start on one day and end on another day, whereas the normal events start and end on the same day. Whole-day, cross-day and normal events are displayed on the timeline.

Calendar 2.0 application implements a type of shrink-and-hide mechanism that shrinks the overlapped events. It also adds transparency to the overlying event. Underlying events are therefore made visible due to the transparency and smaller width of the top ones.

Figure B.10: Representation of conflicting events
External weather service
The active service implemented in the prototype is the weather service. This is accomplished by implementing the most important methods from the abstract class. The Refresh method checks each event attached to the external service by comparing the anticipated weather to the forecasted weather. When a differentiation is detected, the external service automatically updates the status of the event in the database. This in turn generates a notification message when the GetMessages method is invoked. Each location has a file saved in a Dictionary with cities and countries as keys (e.g. “Växjö, Sweden”) and the URIs of the XML files as values. The Refresh method fetches weather forecast information from the XML file corresponding to the key and updates the events attached to the external service. Notification messages returned by GetMessages method are shown in the notification bar in the GUI.

Test
There is a test project (Figure 8.11) containing NUnit test cases for the utilities in the framework, which can be used for regression testing. To run the test cases, some pre-conditions have to be fulfilled (or some expected results have to be changed in the test cases). The pre-conditions are stated in the corresponding test cases.

The prototype is mainly tested using manual testing. In addition, some Selenium test cases are implemented by using NUnit to test executions of reading events.

Test

![Test Structure Diagram]

Figure B.11: The structure of the test projects

Sequence diagrams
This section is to depict sequence diagrams which is vital to the project.
Figure B.12: Manage an event attached to external service

Figure B.13: Get notifications for active events
Figure B.14: Manage an event
Appendix C: Evaluation form
This section includes the evaluation form used for the evaluation of the resulting prototype. Text inside a grey square is notations for each task. Text space for answers is omitted to reduce space.

☐ Male ☐ Female

Age Välj ett objekt.

☐ Private ☐ Work

Default calendar application Klicka här för att ange text.

Using calendar on device(s) ☐ Pad ☐ Mobile ☐ Computer

Nationality Klicka här för att ange text.

Frequency of usage Välj ett objekt.

Area of study/Work Klicka här för att ange text.

Participant nr: Klicka här för att ange text.
Task 1: Week navigation

Scenario
A customer calls and wants to book a meeting at week <week no.> between 8am and 5pm, Monday to Friday.

Task
Your task is to find a suitable time for that meeting to take place.

Questions
- What support did Google Calendar give you to perform the task? (Thoughts from user summarised).
- What support did Calendar 2.0 give you to perform the task? (Thoughts from user summarised).
- Which do you prefer and why?
- Does this feature give you any value in your day to day use of the calendar?

Future work
How can this feature be improved to increase your value of the calendar application?

Reason for evaluation
Since weeks are frequently used by many people, at least in Sweden, it is a good thing to know how much value a week view might give users.

Expected outcome
GC:
- The user navigates through month view.
- The user clicks navigation arrows in week view.
- The user knows about the date of that week and navigates using the dates (Low probability).

C2:
- The user clicks the week view to get to the corresponding week.

Pre-conditions
- A couple of events at the specific day of the event.
- Week should be at least two weeks ahead.

Task 2: One week planning

Scenario
A friend asks you to have a beer with him. He arrives tomorrow and will then be in town for a week. Since he is working during the days, only evenings are feasible meeting times.

Task
Your task it to find a set of suitable times when you can meet with him while he is in town.
Questions

- What support did Google Calendar give you to perform the task? (Thoughts from user summarised).
- What support did Calendar 2.0 give you to perform the task? (Thoughts from user summarised).
- Which do you prefer and why?

Future work

This is a kind of planning issue. Will it give you any value to have a more sophisticated planning tool in the Calendar? If yes, give an example of usage!

Reason for evaluation

A view that is not depending on the static number of days and can start at any day and last for x days ahead has to have some advantage over the normal view. In this task we try to find out if it can have some advantage when viewing a set of days starting at one specific day where the static views do not match that view.

Expected outcome

GC:

- The user changes the customised view to 7 days or more. This requires the user to go to the settings, change the number of customised days, and go back to week view.
- The user will skip through weeks. In the best case scenario (Seen from Google Calendar’s perspective it is a Sunday so the user can see the whole week), that is 1/7 probability, so let’s see too that it is not the case when evaluating. Also have to scroll down.
- The user uses monthly view to get an overview of the week. This we prohibit by creating some tasks for each day so that it will be hard to see. If the user stills do this, the change of orientation should make it harder to perform the task.

C2:

- The user uses the default of 7 days and first clicks the “>” after which he directly finds a time during the evening the coming 7 days.
- The user increases the number of days in the view to 8+.

Pre-conditions

- Calendar should be full most of the evenings of the available days. And events should exist in the days as well.
- The test shouldn’t take place on a Sunday since the week will then be Mon-Sunday in the week view of Google Calendar.

Remark

In the specific case Google Calendar may have the same view as we and the tasks will be identical to perform. We can easily argue that this is the specific case and happens rarely 1/7 or something and should therefore not be a valid evaluation. If our application performs exactly the same as Google Calendar for those specific cases, we can easily argue that our view outperforms Google Calendar.
Task 3: More advanced, two week planning

Scenario
Your boss asks you to carry out a task that will take you about 40 hours. You work between 8 am and 5 pm. Assume work days are Monday to Sunday.

Task
Your task is to find out whether this can be performed within a 2 weeks range or not.

Questions
- What support did Google Calendar give you to perform the task? (Thoughts from user summarised).
- What support did Calendar 2.0 give you to perform the task? (Thoughts from user summarised).
- Which did you prefer and why?

Future work
Do you think it is enough to have 14 days as a maximum or do you prefer having more days? If yes, give a scenario and explain why!

Reason for evaluation
This task will evaluate the advantages of having a view that can cover two weeks. Even if this task is quite similar to the previous one, the week view in Google Calendar should not be enough to carry out this task efficiently. Therefore the slider should have an advantage over the week view. And it should be even more obvious that the dynamic view based on days together with the slider is to be more efficient than any view in Google Calendar.

Expected outcome
GC:
- The user changes the customised view to 2 weeks +.
- The user uses week to perform the task.
C2:
- The user uses the slider to change number of days in view.

Pre-conditions
- The calendar from today and two weeks ahead is 40-60% populated (according to working hours).

Task 4: User specified properties

Scenario
You subscribe to a calendar with lectures for a course you are taking. Two weeks ago, <Day> <Date> you made some notes that are related to that lecture. The lectures name is <Name>.

Task
Your task is to find the notes related to that lecture.

Questions
- What support did Google Calendar give you to perform the task? (Thoughts from user summarised).
- What support did Calendar 2.0 give you to perform the task? (Thoughts from user summarised).
- If you have encountered a similar problem before, i.e. to keep notes attached to events from read-only calendars, how do you normally solve it? If yes, would you prefer to use this approach or your old one?

**Future work**
Can you think of other scenarios where own fields on read-only events could be usable?

**Reason for evaluation**
Keeping personal data attached to read-only events gives the advantage of being able to keep information that you later want to retrieve or information that you might want to retrieve when the event is to happen.

**Expected outcome**

**GC:**
- The user most probably tries to find the event and figures out that there are no own fields on read only events.
- The user already knows that you can’t store information on read-only events.

**C2:**
- The user navigates to the event and finds the notes.

**Pre-conditions**
- Create note on subscribed event for calendar called <name>

**Task 5: Active Event**

**Scenario**
You are going on a picnic on <Day>. Of course you want it to be sunny. If the weather changes to something else you want to be notified so that you can change your plans.

**Task**
Create an event that notifies you when the forecasted weather changes from sun (is something else than sunny). When that happens, you should modify the event, i.e. change date or delete it. This task is done by the observer.

**Questions**
- Do you think the support from Calendar 2.0 was enough to carry out the task or do you think there is something that could be added?
- Will this kind of feature give you any value in a calendar application?

**Future work**
Can you find any other usage of “active events”, i.e. events that are actively connected to respond to changes in an external source other than a Calendar? If yes, give an example and how it could be used.
Reason for evaluation
This feature is new according to existing functionality in Google calendar. To get some feedback of its value and future improvement it is a good idea to include it as a task.

General questions
Q1: Which view do you prefer in a calendar application?
   - The traditional static view containing a predefined set of days and a start day that is set by default?
   - The dynamic view introduced in Calendar 2.0 whereas the number of days in the view can be adjustable and the view starts with a day easily adjustable by you?

Q2: What do you think of the representation of conflict events in Calendar 2.0?
Q3: Overall what did you think of Calendar 2.0 compared to Google Calendar?
Q4: Do you have any other ideas that can be added to increase the value of Calendar 2.0?