
# Software Project Management Plan

Vrije
Universiteit
Brussel

Lars Van Holsbeeke
1.0

**Revision History**

| Version | Date | Description |
|---|---|---|
| **0.1** | 29/10/2013 | Creation of document structure |
| **0.2** | 03/11/2013 | Completion of initial version |
| **0.3** | 14/11/2013 | Adapted to feedback of initial version |
| **0.4** | 02/12/2013 | Adapted to feedback of version 0.3 |
| **1.0** | 12/12/2013 | Updated for delivery, iteration 1 |

# Contents

# 1 Overview

## 1.1 Project Summary

### 1.1.1 Purpose, scope, and objectives

The main purpose of this project is to create a working scheduling webapplication with specific support for mobile devices like smartphones and tablets that enables (authorized) users to query their personal course/final schedule and notifies them about last-minute changes. We will call this application: `Xiast`*(**X**iast **i**s **a** **s**cheduling **t**ool)* More specific requirements can be found in the Software Requirements Specification document.

The main system itself uses the `Wilma` server of the university as back-end and a normal or mobile browser as front-end.

All documents, source code and other artifacts are publicly available on Github. Documents can be found under the `se1-1314/xiast-docs` repository, source code can be found under `se1-1314/xiast` repository.

This academic 3rth bachelor project is part of the course "Software Engineering", taught by dr. R. Van Der Straeten taking place at the "Vrije Universiteit Brussel"

### 1.1.2 Assumptions and constraints

Some constraints involving documentation standards, infrastructure and use of certain technologies have been defined by the client:

**Documentation**

- This document (SPMP) must conform the IEEE 1058-1998 standard
- The SRD, SDD, STD, SQAP and SCMP must also conform their IEEE xxx-1998 standard or a more recent revision of that standard.
- All documents must be written or in Dutch or in English, but not a combination of the two.
- All documents must be available in the PDF format
- At least following documents must be maintained:
  - Software Project Management Plan (SPMP)
  - Software Test Plan (STD)
  - Software Requirements Specification (SRS)

  - Software Design Document (SDD)

- Meeting minutes must be made for all meetings
- An SCMP and an SQMP are not necessary, but all relevant information concerning them must be found in the SPMP.

**Language**

- Only Clojure, Java, JavaScript, HTML, CSS, SQL and corresponding libraries and open-source frameworks
- Only open-source software may be used for both the endproduct and tools
- A particular choice of library, tool, etc. must be motivated by means of reliabilityn, openness and simplicity.
- A library can only be used after agreement with the client and a comparative study of other possible libraries.

**Infrastucture**

- The VUB "Wilma" server must be used as backend for the system.
- The system must work on a browser as frontend.
- The system must work on a mobile browser.

**Other Constraints**

- "Github" must be used as public repository for the code.
- All documents, source code and other artefacts must be publicly available in a structured way.
- The system must have a standard, easy installation procedure.
- The UI must be simple and attractive to use.
- Requirements IDs may never be renumbered.
- All of the code needs to be documentated.
- Test must be written using the "JUnit" framework.
- The system must be modular in design to accomodate extension and replacement of the containing modules.
- The development proces must be iterative with incremental delivery.

### 1.1.3 Project deliverables

The table table below shows code, document and other deliverables with their corresponding deadline: 9 o'clock in the morning on the date shown.

| Date | Deliverable |
|---|---|
| 04/11/2013 | First version of the SPMP |
| 15/11/2013 | First version of documents |
| 18/11/2013 | Data dump: data available for use |
| 13/12/2013 | End of first iteration: delivery of code and documents |
| 18/12/2013 | First presentation |
| 04/03/2014 | End of second iteration: delivery of code and documents |
| 12/03/2014 | Second presentation |
| 15/04/2014 | End of thirth iteration: delivery of code and documents |
| 16/05/2014 | End of fourth iteration: final delivery of code and documents |
| 21/05/2014 | Final presentation |

### 1.1.4 Schedule

Section 5.2 describes the work plan of the project, which contains a detailed description of the work activities with the corresponding teammembers that work on it along with an estimation of time they will need to complete it.

## 1.2 Evolution of the SPMP

This SPMP will be reviewed at least one time a week by the projectmanager. If needed, this document will be updated by the same person. Each (major) update will be logged in the Revision History, to be found at the beginning of this document.

# 2 References

1. SRS: Software Requirements Specifiaction

Anders Deliens
https://github.com/se1-1314/xiast-docs/blob/master/management
/requirements/requirements.md

2. Software Engineering course, VUB

   Catalog number: 1004483BNR
   https://caliweb.cumulus.vub.ac.be/caliweb/?page=course-offer&id=001462
   &anchor=2&target=pr&year=1314&language=en&output=html

3. Wilma backend server

   http://wilma.vub.ac.be

4. Github

   https://github.com

5. JUnit framework

   http://junit.org/

6. Markable.in

   Online document writing tool for the Markdown language. markable.in

7. Iterative and Incremental development model

   Is any combination of both iterative design or iterative method and incremental build model for development. For more information:
   http://en.wikipedia.org/wiki/Iterative_and_incremental_development

8. Agile Software Development

   Topic in the Software Engineering course. Is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. More information on the course slides or http://en.wikipedia.org/wiki/Agile_software_development

9. Boehm's spiral model

   Is a risk-driven process model generator for software projects. Further information: http://en.wikipedia.org/wiki/Spiral_model

# 3 External Stakeholders

1. Ragnhild Van Der Straeten
   Professor of the Software Engineering course. rvdstrae@vub.ac.be
2. Jens Nicolay
   Assistant of the Software Engineering course. jens.nicolay@vub.ac.be
3. Dirk Van Deun
   System administrator of the Wilma backend server. dirk@dinf.vub.ac.be

# 4 Definitions

| Acronym | Declaration |
|---|---|
| **DaM** | Database Manager |
| **DeM** | Design Manager |
| **CM** | Configuration Manager |
| **IEEE** | Institute of Electrical and Electronics Engineers |
| **PM** | Project Manager |
| **RM** | Requirements Manager |
| **QAM** | Quality Assurance Manager |
| **SDD** | Software Design Document |
| **SPMP** | Software Project Magement Plan |
| **SRS** | Software Requirements Specification |
| **STD** | Software Test Plan |
| **SQAP** | Software Quality Assurance Plan |
| **SDP** | Software Documentation Plan |
| **VUB** | Vrije Universiteit Brussel |
| **PDF** | Portable Document Format |
| **UI** | User Interface |
| IDE | Integrated Development Environment |

Other definitions can be found on page 2-3 of the IEEE 1058-1998 standard for Software Project Management Plans

# 5 Project Organisation

## 5.1 External interfaces

Client

In this project the titular of this course, Software Engineering, mrs. R. Van Der Straeten, will together with her assistant, mr. J Nicolay, act as client for the project. This means that all communication involving requirements and design will pass by at least one of them and respectively the Requirements Manager and the Design Leader. All other communication with the client will be handled by the Projectmanager, this includes submitting deliverables: source-code and documents, communication involving presentations, etc.

### 5.1.1 Infrastructure

All communication concerning the available infrastructure: the Wilma back-end server will be handled with the head of infrastructure, mr. D. Van Deun by the web- and databasemanager.

### 5.1.2 External Scheduling Data

Any problems, remarks,... involving the dump of scheduling data on November 18th, 2013 will be communicated to the professor of the course, mrs. R. Van Der Straeten.

## 5.2 Internal Structure

### 5.2.1 Internal Communication

All communication between the teammembers outside meetings must be logged by or the issue tracker on Github or using the internal mailinglist: se1_1314@wilma.vub.ac.be. This is a rule of thumb that must be followed by every teammember. Only if the information to communicate is not important, irrelevant to the other teammembers, does not involve agreements, deadlines, etc. and the urgency of the concerning activities is very low,

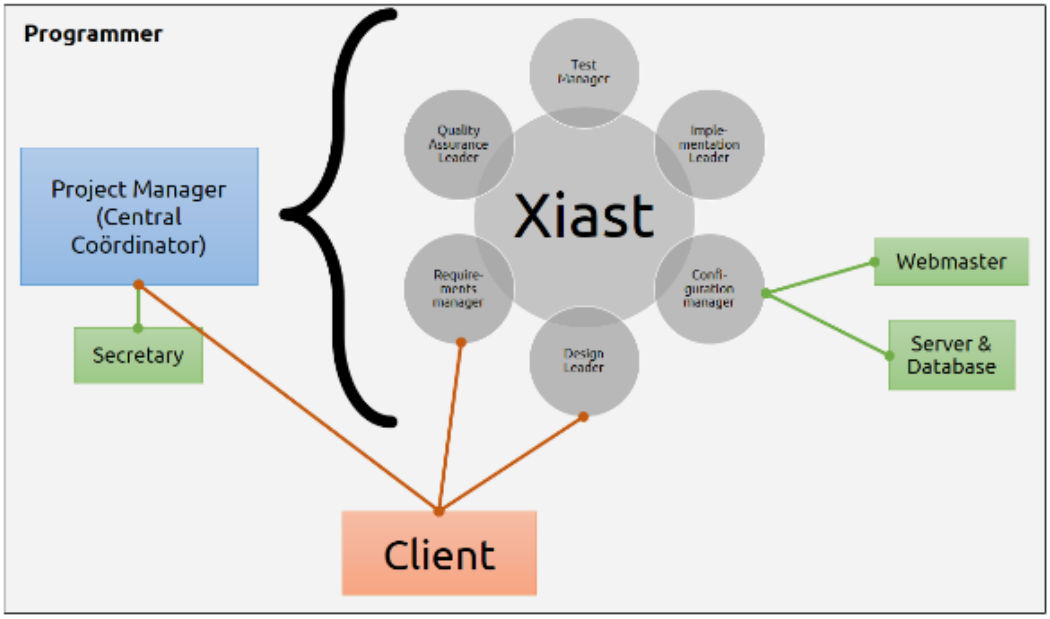



Figure 1: Internal organisation

teammembers can use private mail. In case of urgent problems, problems with another teammember, important matters that need immediate attention, etc. teammembers may use the private mobile phone number of the Projectmanager that has been given to them in the second meeting.

### 5.2.2 Internal Organisation

The chart below shows the internal organisation and flows of information between the actors of the team:

- The Projectmanager acts as a central coordination point for the whole team. This means that he is responsible to solve (personal) issues between teammembers, He also communicates with the client about general project issues concerning planning, progress,... (see Roles and Responsibilities)
- Every manager has the same grade compared to any other manager. This arises from the fact that every teammember fulfills the role of a manager, because of the restricted number of teammembers. In this way every teammember can communicate problems of any kind to the

other teammembers without having to propagate them through a complex hierarchy of chiefs, leaders, managers, etc. "Internal Communication" give more information about how this communication takes place.
- Because of their (semi) overlap with the tasks of their super functions, non-managerial positions (secretary, webmaster and Server & Database manager) will have communicate their issues, reports, etc. to their corresponding manager.
- As already mentioned, only the project manager, design leader and requirements manager can communicate directly to the client.

Roles and Responsibilities

### 5.2.3 Main responsibilities

- Project Manager
  - Creating & providing the SPMP with updates
  - Coordination of the team
  - Contact person for all teammembers
  - Chairman during meetings
  - Creating a weekly meeting agenda on Github
  - Approving decisions taken during meetings
  - Detecting team related problems and solving them
  - Ensuring deadlines are met by all teammembers
  - Ensuring quality of non-code artefacts, created by the teammembers
  - Verifying (together with the secretary) meeting minutes and correcting them if needed
  - Creation of a time-scheme, together with the other teammembers
  - Creation of annotated tags on the Github repository (together with the configuration manager): one for each iteration
  - Creating presentations
- Configuration Manager
  - Creating & providing the SCMP with updates
  - Managing the Github repository for code and documents
  - Managing tools used within the team

  - Providing some documentation concerning the used tools and Git.
  - Ensuring safety and restorability of documents

- Quality Assurance Leader
  - Creation of & providing the STD with updates
  - Optionally creating (and maintaining) an SQAP
  - Quality-based Monitoring of the Software
  - Reviewing source-code: are all required features implemented?
  - Setting up Unit tests

- Requirements Management Leader
  - Creation of & providing the SRS with updates
  - Communicating with client about requirements: p.e. in case of ambiguity, special requests, etc.
  - Determines the priority for each working activity
  - Takes care that activities with higher priority are done first
  - Reporting possible changes to the requirements, made by the client

- Design Leader
  - Creation of & providing the SDD with updates
  - Determining (and managing) the architecture of the system and Database
  - Communicating with the client about the design

- Implementation Leader
  - Managing of the source code
  - Reporting issues concerning the source code on meetings
  - Distributing programming workload to all teammembers
  - Monitoring developers

- Server & Database responsible
  - Regularly updates the website with new information
  - Takes care of communication with the infrastructure manager
  - Manages database, server applications and related services

- Webmaster

  - Maintains the static website, generated by GitHub pages
  - Maintains the project website on which Xiast runs

- Secretary
  - Creates meeting minutes during meetings
  - Maintains and corrects this minutes after eacht meeting

#### 5.2.4 Documentation responsibilities

| Responsible teammember | Document(s) |
|---|---|
| Youssef Boudiba | STD, SQAP |
| Anders Deliens | SRS |
| Adriaan Leijnse | SDD |
| Kwinten Pardon | SDP |
| Nils Van Geele | SCMP |
| Lars Van Holsbeeke | SPMP |

# 6 Managerial Process Plans

## 6.1 Start-up Plan

### 6.1.1 Staffing Plan

H = Function Holder, B = Back-up

| Function/Teammember | Youssef Boudiba | Anders Deliens | Adriaan Leijnse | Kwinten Pardon | Nils Van Geele | Lars Van Holsbeeke |
|---|---|---|---|---|---|---|
| Project Manager | | B | | | | H |
| Configuration Manager | | | | | H | B |
| Quality Assurance Leader | H | | | B | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Requirements Manager | | H | B | | | | |
| Design Leader | | | H | | B | | |
| Implementation Leader | B | | | H | | | |
| Secretary | | B | | | H | | |
| Server & Database | | | B | | H | | |
| Test Manager | H | | | B | | | |
| Webmaster | | | | | H | B | |

### 6.1.2 Project Staff Training Plan

Each teammembers is responsible to become familiar with the technologies, languages, etc. used in the project. It is therefore highly recommended to use information from books, (online) tutorials, other teammembers, etc. to resolve the lack of any foreknowledge. At the beginning of the project, the situation is as follows

- Youssef: Has experience with Java and Git but hasn't any experience with Clojure, Latex and webtechnologies
- Anders: Has programming skills and lots of experience with group projects and LaTeX, but never used Git and Clojure before
- Adriaan: Has lots of experience with Git, Clojure and LaTex but less with webtechnologies
- Kwinten: Is very experienced with webtechnologies, has lots of experience with Git, less with LaTex and none with Clojure
- Nils: Has lots of experience with Git, LateX and webtechnologies, but less with Clojure
- Lars: Has experience with Java and LateX, less with Git and webtechnologies and none with Clojure.

Based on this information it would be very useful for teammembers to follow tutorials concerning the technologies/languages they are not familiar with. This leads to the following situation: *XX = highly recommended, X = recommended, V = not really necessary*

| Training | Youssef | Anders | Adriaan | Kwinten | Nils | Lars |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| Clojure | XX | XX | V | XX | X | XX |
| LaTeX | X | V | V | V | V | V |
| Git | X | XX | V | V | V | XX |
| Webtechnologies | XX | XX | X | V | X | X |

## 6.2 Work Plan

### 6.2.1 Work activities

The table below shows an overview of the different activities in the development process together with the responsible teammember and an estimation of time needed to complete the activity. Rough time estimations were made by the group and are based on the total workload of each package of activities concerning this iteration (iteration 1). This way of estimation has been chosen because the models (Albrecht/IFPUG, Symons/Mark,COSMIC,COCOMO8I, COCOMOII, ...) are made for business software development in the real world (with a real company). We are only students simulating a software company, we don't have the amount of rescources, infrastructure,... a real company has. In this way these models would lead to untrustworthy (time)estimations.

### 6.2.2 Planning per iteration

- **Iteration 1 "Writing web applications in Clojure"**, *December 13th, 2013*
  - Interfaces: A single web interface using dynamically generated static web pages is used.
    - Internationalisation: The interface needs to be displayed in both English and Dutch, depending on user preference.
  - Logging in: Using the VUB's authentication API.
  - Courses: Viewing a list of courses, optionally filtered through a keyword.
  - Schedules: Viewing the entire schedule of a course, logged in student or room.

- **Iteration 2 "Making Clojure and JavaScript work together"**, *March 4th, 2014*
  - Interfaces: Both a mobile and a desktop interface need to be provided.
  - Schedules for whole programs
  - Permission system: Program managers can only change the courses they own, etc.
  - Configuration file driven scheduling algorithm: To make a simple start without introducing too much complexity in the front-end.
  - Program managers can add programs through a web-interface
  - Program managers can add rooms through a web-interface
  - Instructors can change course details of existing courses: E.g. "this course requires an overhead projector", etc.
- **Iteration 3 "Full base functionality"**, *April 15th, 2014*
  - Web interface for program manager business: Scheduling courses, configuring programs, etc.
  - Messaging system for schedule suggestions: Instructors and program managers need to communicate/request schedule changes. These should be easily appliable to the actual schedule by a program manager.
- **Iteration 4 "Extra features and polishing up"**, *May 16th, 2014*
  - E-mail notifications for students about schedule changes

**Time estimations** Time estimations are made in hours (h).

| Activity | Responsible | Est. Time | Documents |
|---|---|---|---|
| Quality Checks | QAM | 5 | (SQAP) |
| Tests | QAM | 15 | STD |
| Requirements management | RM | 35 | SRS |
| Design | DeM | 20 | SDD |
| Implementation | IL, programmers | 50 | source code |

| | | | |
|---|---|---|---|
| Configuration management | CM | 30 | SCMP |
| Team management | PM | 60 | SPMP |
| Training | n.a. | 30 | n.a. |

**Iteration 1**

**Actual performed time** During the development proces, each teammember will log how much time he spends on an activity of the project. This includes time spend on programming, documentation, testing, versioning control, etc. but also time spend on meetings. At every (weekly) meeting, each team member should tell how much time he has spent on which activity.

Performed time has been logged in hours (h).

| Teammember | Function | Perf. Time | Documents |
|---|---|---|---|
| Youssef Boudiba | QAM | 27 | STD, (SQAP) |
| Anders Deliens | RM | 31 | SRS |
| Adriaan Leijnse | DeM | 52 | SDD |
| Kwinten Pardon | IL, programmers | 44 | source code |
| Nils Van Geele | CM | 44 | SCMP |
| Lars Van Holsbeeke | PM | 67 | SPMP |

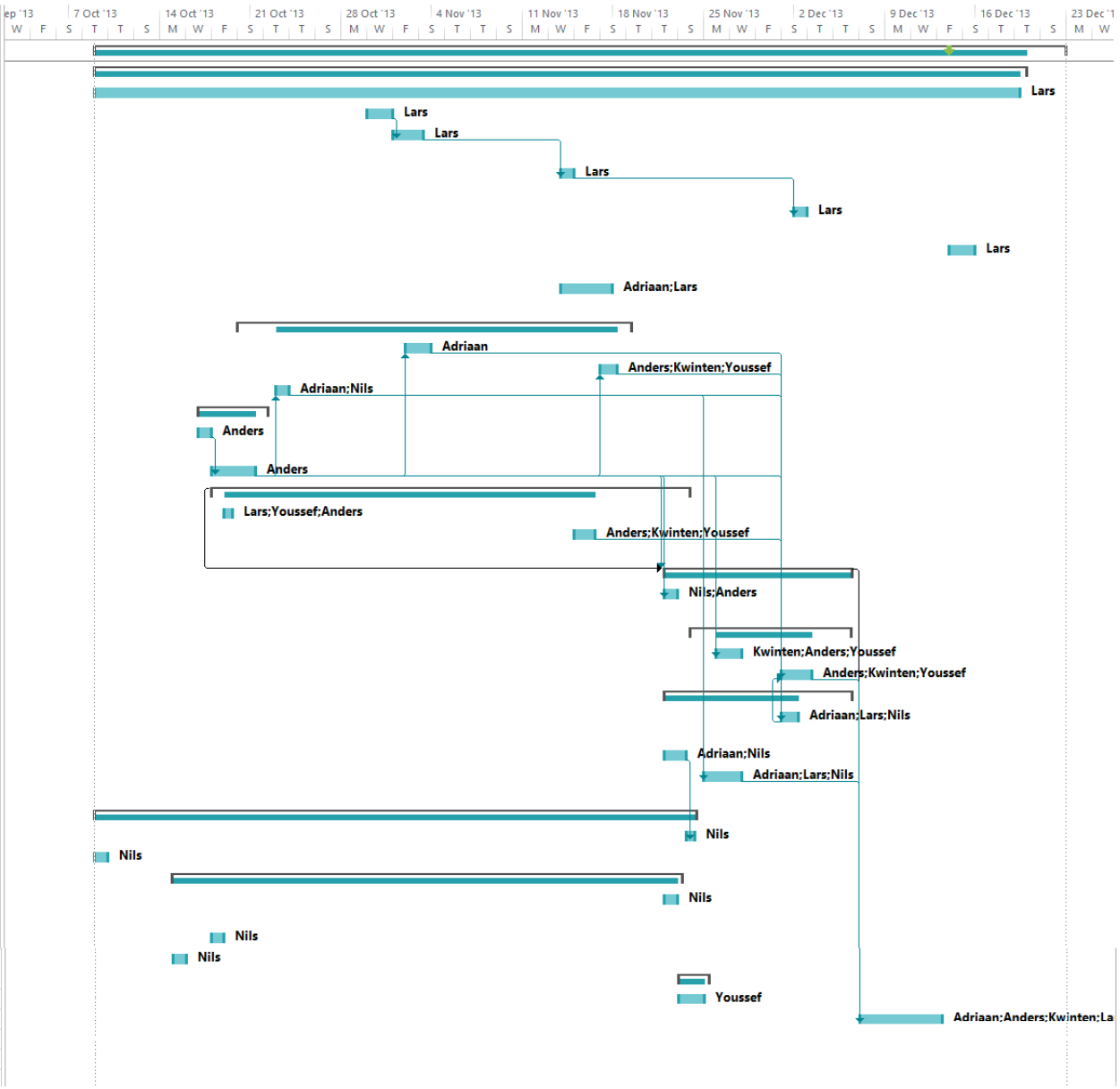
*TOTAL: 265 hours*

### 6.2.3 Schedule allocation

A GANTT chart is used for this. The major part of the activities depends on the release of the SRS, because the latter contains all requirements on which our project is build. The majority of the dependencies between activities are logically derivable from the GANTT-chart.

GANTT CHART

| | Task Name | Duration | Start | Finish | Predecessors | Resource Group |
|---|---|---|---|---|---|---|
| 1 | **Iteration 1** | **93,5 days** | **Wed 9/10/1** | **Sun 22/12/13** | | |
| 2 | **Management** | **89,5 days** | **Wed 9/10/1** | **Thu 19/12/13** | | **Back-end** |
| 3 | Managing Team | 89 days | Wed 9/10/1 | Thu 19/12/13 | | Back-end |
| 4 | Creation SPMP | 2 days | Wed 30/10/1 | Thu 31/10/13 | | Back-end |
| 5 | Updating SPMP: version 0.2 | 3 days | Fri 1/11/13 | Sun 3/11/13 | 4 | Back-end |
| 6 | Updating SPMP: version 0.3 | 1 day | Thu 14/11/13 | Thu 14/11/13 | 5 | Back-end |
| 7 | Updating SPMP: version 0.4 | 1 day | Mon 2/12/13 | Mon 2/12/13 | 6 | Back-end |
| 8 | Creating Presentation | 3 days | Sat 14/12/13 | Sun 15/12/13 | | Back-end |
| 9 | Creating GANTT Chart | 5 days | Thu 14/11/13 | Sun 17/11/13 | | Back-end |
| 10 | **Design** | **39 days** | **Sat 19/10/13** | **Tue 19/11/13** | | **Back-end** |
| 11 | Creating SDD | 3 days | Sat 2/11/13 | Sun 3/11/13 | 16 | Back-end |
| 12 | UI-Mockup | 2 days | Sun 17/11/1 | Mon 18/11/13 | 16 | Front-end |
| 13 | Query-API Design | 1 day | Tue 22/10/1 | Wed 23/10/13 | 16 | Back-end |
| 14 | **Requirements** | **7 days** | **Thu 17/10/1** | **Tue 22/10/13** | | **Front-end** |
| 15 | Analysis of user-requirements | 1 day | Thu 17/10/13 | Thu 17/10/13 | | Front-end |
| 16 | Creating SRS | 5 days | Fri 18/10/13 | Mon 21/10/13 | 15 | Front-end |
| 17 | **Training** | **46 days** | **Fri 18/10/13** | **Sat 23/11/13** | | **Back-end;Front-end** |
| 18 | Git Training | 1 day | Sat 19/10/13 | Sat 19/10/13 | | Back-end;Front-end |
| 19 | Clojure Frontend Training | 2 days | Fri 15/11/13 | Sat 16/11/13 | | Front-end |
| 20 | **Implementation** | **18 days** | **Fri 22/11/13** | **Fri 6/12/13** | **16;17SS** | **Front-end** |
| 21 | Creating issues from requirements | 1 day | Fri 22/11/13 | Fri 22/11/13 | 16 | Back-end;Front-end |
| 22 | **Front-end** | **15 days** | **Sun 24/11/1** | **Fri 6/12/13** | | **Front-end** |
| 23 | Creating Mock-up | 2 days | Tue 26/11/1 | Wed 27/11/13 | 16 | Front-end |
| 24 | UI-Implementatio | 3 days | Sun 1/12/13 | Tue 3/12/13 | 26SS;13;19;12;16 | Front-end |
| 25 | **Back-end** | **18 days** | **Fri 22/11/13** | **Fri 6/12/13** | | **Back-end** |
| 26 | Authentication (simple) | 2 days | Sun 1/12/13 | Mon 2/12/13 | 16;11 | Back-end |
| 27 | Database Design | 2 days | Fri 22/11/13 | Sat 23/11/13 | | Back-end |
| 28 | Query API implementation | 3 days | Mon 25/11/13 | Wed 27/11/13 | 13 | Back-end |
| 29 | **Configuration** | **58 days** | **Wed 9/10/1** | **Sun 24/11/13** | | **Back-end** |
| 30 | Database Import | 1 day | Sat 23/11/13 | Sun 24/11/13 | 27 | Back-end |
| 31 | Creation SCMP | 1 day | Wed 9/10/1 | Wed 9/10/13 | | Back-end |
| 32 | **Server** | **48 days** | **Tue 15/10/1** | **Sat 23/11/13** | | **Back-end** |
| 33 | Infrastructure configuration | 1 day | Fri 22/11/13 | Fri 22/11/13 | | Back-end |
| 34 | Website | 1 day | Fri 18/10/13 | Fri 18/10/13 | | Back-end |
| 35 | Repository | 1 day | Tue 15/10/1 | Tue 15/10/13 | | Back-end |
| 36 | **Quality-Assurance** | **4 days** | **Sat 23/11/13** | **Mon 25/11/13** | | **Front-end** |
| 37 | Creation SQAP | 3 days | Sat 23/11/13 | Sun 24/11/13 | | Front-end |
| 38 | **Testing** | **8 days** | **Sat 7/12/13** | **Fri 13/12/13** | **24;28;20** | **Back-end;Front-end** |



### 6.2.4 Resource allocation

An overview of rescources that will be used can be found in the table below

| Rescource | Activities |
|---|---|
| Wilma backend server | Application backend; Hosting of the static website |
| Microsoft Project | Project Management (tool) |
| Microsoft PowerPoint | Presentations |
| Markable | Writing documents in the Markdown language |
| Github | Versioning Control System |
| Smartphone (Android) | Testing mobile version of the tool |

## 6.3 Control Plan

### 6.3.1 Requirements control plan

Possible changes of requirements will always be communicated between the requirements manager and the client. When a change occurs, the requirements manager puts an new topic on the agenda of the next teammeeting and updates the SRS.

### 6.3.2 Schedule control plan

Problems involving scheduling, deadlines, etc. will be discussed during the weekly meeting. Each teammember is responsible to keep track of his deadlines, and will report (at the weekly meeting) what he has done on which activity during the last week. The projectmanager himself will keep track of the global planning by using these reports and make adjustments to the planning and/or activity if needed. If it seems that one of the teammembers won't make the deadline, one or more other teammembers can jump in on the activity concerned. This is highly appreciated.

### 6.3.3 Quality control plan

All code and documentation will be periodically checked by the Quality Assurance Manager and before the end of each iteration. First he reports (if

needed) to the concerning person. If any severe (quality based) problems are detected, he will report also them at the weekly meeting.

### 6.3.4 Reporting plan

Using the SPMP, SCMP, STD and SDD, the status of the project will be reported to external entities (p.e. the client). All this documents are free to be read by anybody on our Github repository. It can be reached and downloaded by using our static website on http://wilma.vub.ac.be/~se1_1314

## 6.4 Risk management plan

This list will be extended in future versions of this document All estimations are on a scale from 0 to 10.

1. One of the teammembers is sick or leaves
    - Probability: medium
    - Impact: high
    - Priority: high
    - Cost of solution: high
    - Solution: Teammember with corresponding back-up function takes over.
    - Target completion date: n.a.
    - Responsible: Project Manager
2. Bad communication between teammembers
    - Probability: medium
    - Impact: medium
    - Priority: high
    - Cost of solution: low
    - Solution: Don't use too much private communication, use the mailinglist. The issue tracker on Github must be up-to-date at all times.
    - Target completion date: n.a.
    - Responsible: Project Manager
3. Not meeting deadlines

- Probability: medium
- Impact: high
- Priority: high
- Cost of solution: low
- Solution: Keeping track of progress made using Github functionality, weekly progress reports of teammembers.
- Target completion date: n.a.
- Responsible: Project Manager

4. Lack of software quality

   - Probability: low
   - Impact: low
   - Priority: low
   - Cost of solution: medium
   - Solution: Periodically quality checks, tests,... Reporting them to the weekly meeting. QAM gives recommendations to the teammembers on the weekly meeting and by using the mailing list. Making and resolving issues on the Github issue tracker.
   - Target completion date: n.a.
   - Responsible: Quality Assurance Manager

5. Misunderstandings between client and team

   - Probability: low
   - Impact: high
   - Priority: high
   - Cost of solution: medium
   - Solution: Regular meetings with the client to check if product meets expectations
   - Target completion date: n.a.
   - Responsible: Requirements Manager

6. Conflicts between teammembers

   - Probability: low
   - Impact: high
   - Priority: high
   - Cost of solution: high

- Solution: Negotiation between the teammembers involved, together with the projectmanager.
- Target completion date: n.a.
- Responsible: Project Manager

7. Abrupt changes in requirements

   - Probability: low
   - Impact: high
   - Priority: high
   - Cost of solution: medium (depends)
   - Solution: Using the modularity of the software product to implement as easily and efficiently as possible the changes. Prevention by involving the client in the development process.
   - Target completion date: n.a.
   - Responsible: Requirements manager, Implementation Leader

8. Wrong interpretation of requirements by the team

   - Probability: medium
   - Impact: high
   - Priority: high
   - Cost of Solution: medium (depends)
   - Solution: Using the modularity of the software product to correct as easily and efficiently as possible the requirements that were misunderstood
   - Target completion date: n.a.
   - Responsible: Requirements Manager, Implementation Leader

9. Apache server goes offline on Wilma

   - Probability: low
   - Impact: medium
   - Priority: high
   - Cost of Solution: low
   - Solution: Changing the url in the configuration of the application
   - Target completion date: n.a.
   - Responsible: Server Manager

10. Database server goes offline

- Probability: low
- Impact: high
- Priority: high
- Cost of Solution: low
- Solution: Changing the configuration of the application (dataloss of a part of the database is possible). The cost of this solution is low because back-ups of the database are taken every hour, using a back-upserver of a teammember
- Target completion date: n.a.
- Responsible: Server Manager

11. Back-end server goes (temporarly) down

    - Probability: low
    - Impact: high
    - Priority: high
    - Cost of solution: low
    - Solution: Using a mirror server: Aphrodite
    - Target completion date: n.a.
    - Responsible: Server Manager

### 6.5 Closeout plan

Not of any importance to this project.

# 7 Technical Process Plan

## 7.1 Process model

We will be using the Iterative and Incremental development model with some ideas of Agile Software Development, which is based on this model. This method has been chosen firstly because of the agenda of the project which consists of an incremental delivery based on four iterations. Secondly, it has been chosen for its simplicity and added value: we focus on a working application per iteration which can than be discussed with the client. In this way we open ourselves up to requirements changes which will be given to us by the client at the end of each iteration. This results in a continuous delivery of valuable software, one of the key principles of agile development. The figure

below shows (Boehm's) spiral model, which will be used as development process model.

- One iteration consists of four phases:
  - Determination of objectives: In this phase changes in requirements will be determined and introduced in the SRS
  - Identification (and resolving) risks: This phase involves the identification of possible risks caused by changed requirements: these will be discussed in the weekly meetings
  - Developing and testing: Using the changed/extended design of the previous iteration, new and changed functionality will be implemented by the developers. The coordination of this phase is being done by the implementation leader, software quality assurance manager, design manager and test manager
  - Planning of the next iteration: Releasing the new version of the product is the first thing to be done in this iteration. After this, the next iteration will be planned by the project manager.

## 7.2 Methods, tools and techniques

- Github will be used as
  - communication tool for documents
  - versioning control system for source code
- Clojure will be used as programming language
  - Back-end: to generate schedules and provide an API to login/logout and to view and modify schedules
  - Front-end: to query schedules using the back-end api.
- JavaScript and JQuery to create dynamic webpages, used to view/update
- Every teammember is free to use his preferred IDE during the implementation process
- A MySQL database will be used as backend database on Wilma. It will be populated with course schedule data.

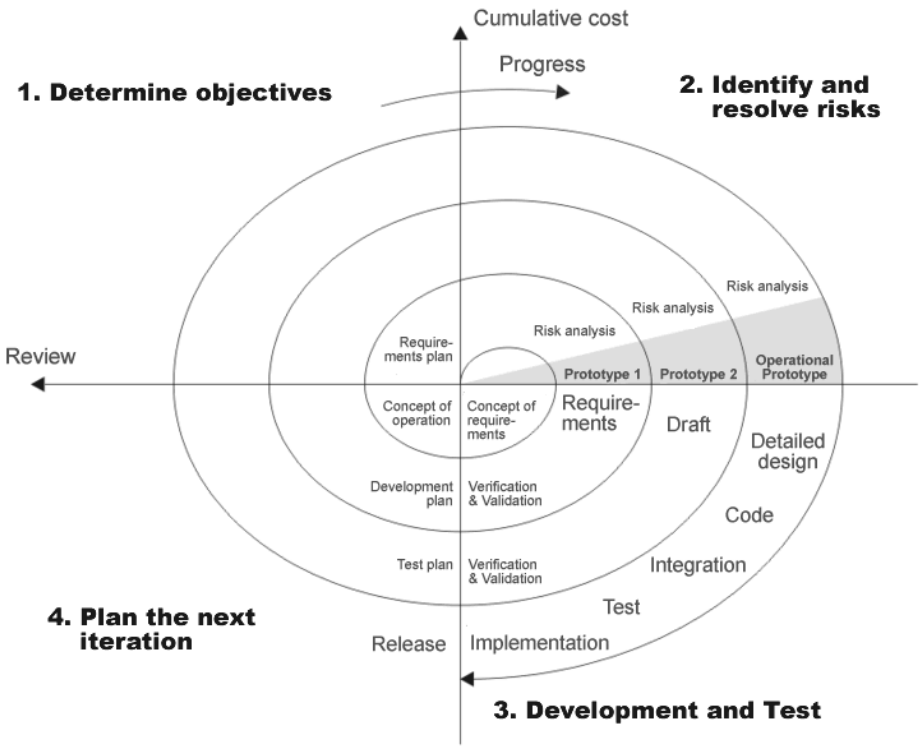



Figure 2: Boehm's Spiral

- Markdown will be used to write documents because of its perfect integration with Github. When documents must be delivered, Pandoc will be used to convert the Markdown documents to the 'paper-compatible' LaTeX language from which pdf files are generated.
- Compojure and Ring are used as the main webframeworks for Clojure.

# 8 Supporting Process Plans

## 8.1 Software Configuration Management Plan (SCMP)

### 8.1.1 Introduction

In this document we will describe the workflow that needs to be followed for the Xiast project. We will also talk about the two **Git** repositories that will be used. This document will also be included in an adapted form in the SPMP.

### 8.1.2 The repositories

For this project we will be using two distinct Git repositories, each with their own specific purpose.

**xiast-docs** The `xiast-docs` repository will be used to hold all documents related to the project. This includes reports made during meetings and all plans made by the different leaders and managers.

Every manager has his own directory inside the `management` directory. In this directory, the manager can put all files related to his work and reports. If a manager wants to make a change to the main directory structure, he is free to do so but must contact the configuration manager about it. By doing this the software configuration manager can keep the directory structure in the SCMP up-to-date.

The current directory structure, which may be subject to change, is:

- /
  - **management** directory containing all official documentation
    - **configuration** concerning project configuration
    - **design** concerning software design

        * **implementation** concerning implementation
        * **project** concerning project planning and management
        * **quality** concerning software quality and testing
        * **requirements** concerning all software requirements
    - **meetings** contains all agendas and reports of meetings
    - **manuals** contains all files related to the Xiast manuals
    - **templates** contains templates for deliverables

**xiast** The `xiast` repository is the main repository that contains all the code for both the server and the Xiast website. The directory structure is that of a basic Leiningen project:

- /
    - **resources**
        * **dictionaries** directory containing translates strings for internationalisation
        * **public** directory containing public files such as images, CSS stylesheets, ...
        * **templates** directory containing all website templates
    - **src** directory containing the source files
    - **test** directory containing all tests

The `xiast` repository is for code and related resources (graphics, sound, SQL, ...) only.

### 8.1.3 Tracking issues

With GitHub's issue tracker, which can be found here[1], we can create so-called *issues*. An issue can be anything from a bug, request for implementation or suggestion, goal, ... By using this tool we can achieve a workflow that will make it easier for both us and others to track the progress of the project.

When all requirements of the project are known, we will start splitting up the requirements in one or more smaller "tasks" that need to be finished in order to implement the requirement. These issues can then be bundled into

[1] https://github.com/se1-1314/xiast/issues

a *milestone* which denotes the requirement or part of the requirement that needs to be implemented. Milestones feature progress trackers which again makes it really easy for people to see how much still needs to be done for a requirement.

For example: "Functioning user system" can be a milestone, with issues such as "User registration" and "User login", or "Access control" with "Assigning user rights" and "Rights checking" as some of the issues.

As mentioned earlier, the issue tracker can and must be used for more than just tracking progress of the requirements. Issues will be made to report bugs and propose fixes, to propose enhancements or request extra functionality, to report on critical errors and so on. Every issue features a comment section which makes it easy for team members to discuss requests, bugs, and so on.

It is important that the issue tracker is kept clean at all times. What this means is that all issues and milestones have proper descriptions and descriptive titles, issues that are finished must be closed (see workflow), when one team member takes over an issue from another team member this must be updated on the tracker, ... and so on.

The usage of the issue tracker, when done properly, causes less overhead on the project because we do not need to maintain our own tracker or website.

### 8.1.4 Git workflow

When working with Git, there are multiple workflows possible. Considiring the size of the team and project, we will be using the so called **feature branching workflow**. This workflow allows us to tightly integrate the issue tracker into our project.

**Branches** There will always be one central branch: the **master** branch. This branch is the actual "master copy" of our project and should contain preferably only working code.

Whenever a team member wants to implement changes, he must first branch the master branch into a new one and commit all his changes to the newly created branch. This branch, which is created locally, needs to be published to GitHub. This doesn't only make it easier to track changes on the branch, but also makes it easier to transfer the work to someone else.

Because some issues have a really small workload and can be solved in a single commit, it's possible to solve multiple issues on one single branch instead of creating a branch for every issue.

To keep things uniform, branch names must be in lower case only and use dashes instead of spaces.

**Merging** After the work has been done and the issue is implemented, the working branch needs to be merged into the `master` branch. It is mainly the team member's responsibility to avoid merge conflicts!

Conflicts can be avoided more easily by periodically pulling and merging the `master` branch into the working branch, and not vice versa. By doing this regulary, conflicts that will occur will be smaller than they would be if we didn't merge at all.

If and only if all work is done and the working branch is conflict free and the head of the branch is pushed to GitHub, steps can be taken to merge the working branch into the `master` branch. This is done by making a **pull request**.

**Pull request** When making a pull request, select the `master` branch as the base branch and the working branch as the compare branch. If the branch that is being merged closes one or more issues, they should be referenced in the description. If done correctly, like described in this guide[2], the issues can be automatically closed when the pull request gets accepted.

After making the pull request, it is the (backup) implementation manager's responsibility to accept or deny the request. The first step is to review the code by testing it locally to see if it actually works. If he so chooses, he can also simulate the merging locally to see if it works with previous accepted pull requests.

If the code works and there are no conflicts, the pull request can be accepted and close. After this, the appropriate issue should be closed too and a comment with a link to the pull request should be made. To avoid "branch pollution" the working branch can safely be removed from the repository by GitHub after accepting the pull request.

If conflicts arise during merging because of for instance earlier accepted pull requests, the implementation manager can choose to fix the conflicts himself manually or just denying and closing the pull request. The second option is advised. In this case, the team member must fix his code and make a new pull request. The same is valid for when there are problems with the code itself.

[2]https://help.github.com/articles/closing-issues-via-commit-messages

**Assigning and reassigning issues** To keep track of who is working on what and to avoid double work, members should assign themselves or others to issues. This can be done on the issue tracker itself.

If a team member is stuck, the member is free to transfer the work to someone else. But he must not forget to reassign the issue to the other person

**docs** For the `xiast-docs` repository, we don't need to use the feature branch workflow, we can just use a **centralized workflow**. This means we won't use branching and we will just commit to the master branch.

### 8.1.5 Project website

One of the requirements of the project is a website that can be used to track the progress of the project. Instead of focussing on implementing our own system, we will make full use of the tools GitHub is providing us.

The website can be reached at `http://se1-1314.github.io/xiast`. This page, which is generated using GitHub Pages, contains information about the project, including download links for the source code and links to various documents.

GitHub Pages is in essence a static website generator. This means that, given the contents of the site, it only needs to generate all the HTML and related files once. This not only makes it faster, but also a lot easier.

The automatic page generator can be found on the settings page of the project. The files it generates are located in the `gh-pages` branch of the `xiast` repository. Commiting to this branch thus allows us to edit the page manually, although it is highly advised to keep using the automatic page generator and edit the content of the site using markdown.

To keep track of the project's status however, we will use the built-in issue tracker of GitHub.

## 8.2 Software Quality Assurance Plan (SQAP)

The Software Quality Assurance Plan (SQAP) is based on the IEEE standard for software quality assurance plans (730-2002). Since almost all information can be found in the SPMP, we will not cover every topics proposed by the standard.

### 8.2.1 Purpose

This SQAP's objective is to ensure that the project does not deviate from its requirements and that a certain level of quality is maintained. By defining several methods and guidelines we ensure that the development of the project proceeds smoothly while keeping the quality high.

### 8.2.2 Tasks

The main tasks of the QAM consist of:

- Writing and updating of the SQAP.
- Writing and updating of the STD.
- Checking whether or not the delivered documents confirm to their purpose.
- Checking if all documents are consistent and follows established guidelines (coding style, use of template for all documents, etc.).

### 8.2.3 Standards, practices, conventions and metrics

**Documentation Standards** All documents must be based on IEEE standards and follow a certain template.

All documentation and source code must be written in English (except meetings reports that will be written in Dutch).

**Coding (and comment) Standards** The code documentation must follow the SDP.

**Testing standards** Testing of code must follow steps of to the STD.

**Metrics** During random checks, the quality of the project will be measured by using following metrics:

- Test Pass Rate.
- Numbers of bugs found per tests.
- Time to fix bug/close issue.

Exact numbers will be communicated later.

**Conventions** We'll organize weekly meetings to resume the current state of the project and to discuss the evolution of it.

### 8.2.4 Test

All the information about testing methodology, procedures and execution can be found in the STD.

### 8.2.5 Problem reporting and corrective actions

**Documents:**

Problems concerning structure and content of documents are communicated to the author. Small mistakes (such as spelling mistakes, etc.), are also informed to the author.

**Source Code:**

Whenever a member does not agree with the implementation of a feature or a small function, an issue will be opened and the code will be reviewed by the group.

### 8.2.6 Tools

See reference.

### 8.2.7 Media Control

Standards described in the SCMP must be followed by the whole team and controlled by the QAM.

### 8.2.8 Supplier Control

Software provide by an external provider must undergo a series of tests and must conform to the requirements before integrating the project.

### 8.2.9 Records collection, maintenance and retenetion

All documents are stored remotely at the gitHub repository (https://github.com/se1-1314) and locally.

### 8.2.10 Training

The project will require certain knowledge of tools and languages. To prevent team members from getting lost during the project, following tools and languages must be mastered by the whole team:

- HTML
- Clojure
- Git
- Java (and JUNIT)

# 9 Additional Plans

Following documents play also a role of importance in this project: SRS, SDD They will be delivered Friday 15th, 2013: deadline for the other documents.