Software Project Management Plan
Version 0.3

Lars Van Holsbeeke
Software Engineering 2013-2014 Group 1

November 14, 2013

Revision History

Version	Date	Description
0.1	29/10/2013	Creation of document structure
0.2	03/10/2013	Completion of initial version
0.3	14/11/2013	Adapted to feedback of initial version

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 SRS (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 xiast-docs, source code can be found under xiast.

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 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/

9. Markable.in

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

10. 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

11. 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

12. Boehm's spiral model

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

3 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/STP	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

4 Project Organisation

4.1 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.

4.1.1 Infrastructure

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

4.1.2 External Scheduling Data

Any problems, remarks,... involving the dump of scheduling data on November 18th, 2013 will be communicated to the infrastructure manager, mr. D. Van Deun.

4.2 Internal Structure

4.2.1 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 thum that must be followed by the teammembers. Only if the information to communicate is such unimportant, irrelevant to the other teammembers, does not involve agreements, deadlines, etc. and the urgency of the concerning activities is very low, 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.

4.2.2 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, he also communicates with the client (see Client). Communicationflows with the client are colored red.

4.3 Roles and 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: one for each iteration
- 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 STP with updates
  - Optionally creating (and maintaining) an SQAP
  - Quality-based Monitoring of the Software
  - Reviewing source-code: are all required features implemented?
  - Setting up JUnit 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 Manager
  - Regularly updates the website with new information
  - Takes care of communication with the infrastructure manager
  - Manages database, server applications and related services

5 Managerial Process Plans

5.1 Start-up Plan

5.1.1 5.1.2 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	H	B				
Server Manager			B		H	

5.2 Work Plan

5.2.1 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. The estimated time may differ from the actual performed time

Activity	Responsible	Estimated Time	Documents
Team management	PM		SPMP

Configuration management	CM	SCMP
Quality Checks	QAM	n.a.
Requirements management	RM	SRS
Design	DeM	SDD
Tests	QAM	STP
Implementation	IL, programmers	source code

Please note that an estimation of time is not yet made in this version of the SPMP. One reason for this is lack of experience. Nevertheless will this estimation be made at the next teammeeting.

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, with a clear separation between managing and coding.

5.2.2 Schedule allocation

A GANTT chart will be used for this. It will be made at the next teammeeting when a License for Microsoft Project 2013 has been obtained.

5.2.3 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

5.3 Control Plan

5.3.1 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.

5.3.2 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.

5.3.3 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.

5.3.4 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

5.4 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: 3
    - Impact: 6
    - Priority: 9
    - Cost of solution: 8
    - Solution: Teammember with corresponding back-up function takes over.
    - Target completion date: n.a.
    - Responsible: Project Manager
2. Bad communication between teammembers
    - Probability: 5
    - Impact: 6
    - Priority: 8
    - Cost of solution: 4
    - 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: 5
    - Impact: 10
    - Priority: 6
    - Cost of solution: 5
    - 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: 3
    - Impact: 3
    - Priority: 2
    - Cost of solution: 5
    - 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: 3
   - Impact: 8
   - Priority: 8
   - Cost of solution: 5
   - Solution: Regular meetings with the client to check if product meets expectations
   - Target completion date: n.a.
   - Responsible: Requirements Manager
6. Not enough knowledge concerning the used programming language (p.e. Clojure, JavaScript, HTML5,...)
   - Probability: 10
   - Impact: 8
   - Priority: 7
   - Cost of solution: 7
   - Solution: Watching tutorials, asking teammembers that know the language for help
   - Target completion date: end of 1st iteration (Clojure basics), end of 2nd iteration (JavaScript, HTML5)
   - Responsible: Design Manager, Implementation Leader
7. Back-end server goes (temporarly) down
   - Probability: 2
   - Impact: 6
   - Priority: 7
   - Cost of solution: 2
   - Solution: Using a mirror server: Aphrodite
   - Target completion date: n.a.
   - Responsible: Infrastructure Manager, Configuration Manager
8. Github Versioning Control System goes down
   - Probability: 1
   - Impact: 6
   - Priority: 7
   - Cost of solution: 2
   - Solution: Using the backup server (Aphrodite) running Gitlab
   - Target completion date: n.a.
   - Responsible: Configuration Manager
9. Conflicts between teammembers
   - Probability: 3
   - Impact: 8
   - Priority: 8
   - Cost of solution: 7
   - Solution: Negotiation between the teammembers involved together with the project-manager.
   - Target completion date: n.a.
   - Responsible: Project Manager
10. Abrupt changes in requirements
    - Probability: 2
    - Impact: 7
    - Priority: 8

- Cost of solution: 6
- 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

11. Client cancels the project

    - Probability: 1
    - Impact: 10
    - Priority: 10
    - Cost of Solution: 10
    - Solution: Closing down the project after double checking/negotiating with the client
    - Target completion date: n.a.
    - Responsible: Project Manager

12. Wrong interpretation of requirements by the team

    - Probability: 5
    - Impact: 7
    - Priority: 8
    - Cost of Solution: 6
    - 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

5.5 Closeout plan

Not of any importance for this project.

6 Technical Process Plan

6.1 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.

6.2 Methods, tools and techniques

At the moment of writing, the programming language has not been chosen yet and will therefore not be mentioned in this version of the SPMP. Therefore, some items will be added to the list below in future versions of the SPMP.

- Github will be used as
  - communication tool for documents
  - versioning control system for source code
- Eclipse will be used as IDE during the implementation process.
- A MySQL database will be used as backend database on Wilma. It will be populated with course schedule data.

7 Supporting Process Plans

7.1 Software Configuration Management Plan (SCMP)

7.1.1 Software Configuration Management Plan

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.

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 directory structure, for instance a new folder inside the management directory, an issue (see below) should be created and closed by the software configuration manager. By doing this the software configuration manager can keep the directory structure in the SCMP up-to-date. Issues should not be created for structural changes inside one's own directory.

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

xiast The `xiast` repository is the main repository that will contain all the code for both the server and the Xiast website. The current directory structure, which again may be subject to change, is:

- /

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

Tracking issues With GitHub's issue tracker, which can be found at `https://github.com/se1-1314/xiast/issues`, 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.

First of all, when all requirements of the project are known and we know more about the design of the software, we will split up every requirement in one or more smaller "tasks" that need to be finished in order to implement the requirement. These issues can then be bundled into a *milestone* which denotes 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.

Furthermore, if bugs are found during testing or actual use of the application, issues can be made for these bugs which will be assigned to a special milestone exclusively for bug fixing.

Requests for implementation and suggestions can also be made through the creation of issues. If a request or suggestion gets denied, we can just simply close the issue (issues can always be reopened). If the request or suggestion gets accepted, the issue needs to be assigned to a milestone specific for this issue.

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.

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 an issue needs to be implemented, it first needs to be assigned to a team member who will then be responsible for the implementation of said issue. The member will then locally, after pulling the repository first, create a new branch from the `master` branch and name it after the issue. He will then checkout the newly made branch, from now on the working branch, and publish it to GitHub.

All work related to implementing the feature will then be done on the working branch and that branch only. Regulary pushing commits made in the working branch is a must.

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 copmare branch. In the description of a pull request a link to the appropriate issue of the working branch should be included.

After making the pull request, it is the 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.

Reassigning issues When a member is assigned to an issue and can't implement or solve it because of reasons, the issue can be reassigned to another team member. In this case, the previous member should make sure the latest commits on the working branch are pushed to GitHub. The new member can then easily pick up the previous member's work.

Updating the issue on the tracker must not be forgotten either!

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.

Tagging At the end of every iteration of the project, the project manager is responsible to make an *annotated tag* for both repositories. The names of the tag should follow the following naming convention: `iteration-{i}.{e}` where `{i}` is the current iteration and `{e}` stand for last minute edits after making the tag starting with 0. This leaves room for a small amount of error.

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.

7.2 Verification and Validation Plan (STD)

This plan will be delivered Friday 15th 2013, 2013: deadline for the other documents.

7.3 SoftwareDocumentation Plan (SDP)

This plan will be delivered Friday 15th, 2013: deadline for the other documents.

7.4 Software Quality Assurance Plan (SQAP)

No seperate plan required for this project. All relevant information concerning this plan can be found in this document, the SPMP. The QAM is responsible for this.

7.5 Problem Resolution Plan

This plan will be delivered Friday 15th, 2013: deadline for the other documents.

8 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.