Software Project Management Plan (SPMP)

Software Engineering and Project

University of Adelaide

Prospector Sea Floor Mapping System

Navdeep Singh (1660360)

Liang Yuan (1679380)

Zeqi Fu (1680895)

Tao Zhang (1680974)

Lili Wu (1683229)

Yi Lin (1682781)

Yann Frizenschaf (1162562)

Semester 2, 2016

Contents

Date	Name	Student ID	Updates	Version
23th Aug	Navdeep Singh	1660360	Added the basic structure of draft	0.1
25th Aug	Navdeep Singh	1660360	Updated the project deliverables	0.2
29th Aug	Lili Wu	1683229	Updated the definitions and glossary	0.3
29th Aug	Yi Lin	1682781	Added the risk management plan	0.4
30th Aug	Yi Lin	1682781	Added the quality assurance plan	0.5
30th Aug	Yi Lin	1682781	Updated the risk management plan	0.6
30th Aug	Liang Yuan	1679380	Added the configuration management plan	0.7
1st Sep	Yi Lin	1682781	Update the quality assurance plan	0.8
3th Sep	Liang Yuan	1679380	Updated the configuration management plan	0.9
4th Sep	Yann Frizenschaf	1162562	Reviewed all the content and added suggestions on revision	0.10
5th Aug	Lili Wu	1683229	Updated the assumptions and constraints	0.11
5th Aug	Lili Wu	1683229	Updated the documentation preparation and versioning	0.12
6th Sep	Yann Frizenschaf	1162562	Updated the process model section and fixed the format problems	0.13
6th Sep	Yann Frizenschaf	1162562	Updated the risk management plan	0.14
6th Sep	Yann Frizenschaf	1162562	Updated the milestone section	0.14
6th Sep	Yi Lin	1682781	Updated the revision history	0.15
6th Sep	Yann Frizenschaf	1162562	draft Release	1.0

Table 1: Revision History

1 Introduction

1.1 Purpose

The purpose of this document is to detail the Software Project Management Plan for the Prospector SFM, developed for SeaFaults. It contains the details of the organisation of the project from a process and resources perspective, elicitation and mitigation of potential risks, a work plan comprising activities, milestones and proposed schedule, and plans for configuration management and quality assurance. As a whole, this document represents a detailed view of the processes and artefacts required to deliver the functionality specified in the associated SRS [4], and to prove that the aforementioned functionality has, in fact, been delivered.

1.2 Scope

This document defines the management plan for the software component of the SFM system only. While hardware considerations are touched on in some sections, the hardware to be used has been defined and consideration of this (beyond its effect on the requirements of the software system) are beyond the scope of both this document and the project as a whole.

1.3 Assumptions and constraints

Assumptions:

- There will be adequate staff available to come out the movement, vision and mapping functions.

- The staff doing movements will accomplish the movement functions manually(forward, reverse and rotation).

- The staff doing movements will also complete the obstacle detecting and avoidance.

- The staff doing vision part will read the values of lighting sensor, and feed the interpreting to both movements and mapping operations.

- The staff doing mapping will export map data based on both movement and vision systems.

- The staff doing GUI will enable control and get visual feedback.

Constraints:

> - Since not all of the staff are good at java, we need more time to complete all the core functions than we expected.

> - The accuracy of sensors from LeJOS EV3 materials are not strictly limited, so the staff will have to measure them and make it as precise as possible to maintain the highest quality.

> - Based on potential changes of requirements, the functions would be modified along with project-processing.

1.4 Project deliverables

The major deliverables for the project have been defined by the client (SeaFaults) and are enumerated below, and due dates for each are specified in Section 4.2:

1. **Software Requirements Specification:** Defines the requirements for the software component of the SFM system, as well as detailing the assumptions inherent in and constraints upon those requirements

2. **Software Project Management Plan:** Defines the technical and managerial Processes necessary for the development and delivery of the SFM system (this document).

3. **Software Design Document:** Describes the design goals of the SFM system, as well as trade-offs made between design goals, software control implementations and systematic constraints.

4. **Testing Report:** Contains the details of all planned an conducted tests of the delivered system, and the outcomes of those tests. Includes both functional tests (system demonstration) and automated tests (i.e. unit tests).

5. **User Manual:** A detailed set of instructions for the SFM system's end user(s). Includes all instructions necessary to perform all delivered functions of the system, and assumes the reader has basic knowledge of computer/GUI use, but no detailed knowledge of the inner workings of the system.

6. **Milestone Software Demonstration 1:** A demonstration of the functionality milestones negotiated with the client to be delivered by an agreed date for milestone 1.

7. **Milestone Software Demonstration 2:** A demonstration of the functionality milestones negotiated with the client to be delivered by an agreed date

for milestone 2.

8. **As-built Software Demonstration:** The major testing event required for the SFM project. Demonstrates to the client all features of the system as-built in a single demonstration session.

1.5   Evolution of the plan

As per the agile project management philosophy detailed in Section 2.1, the details of the plan contained within this document are subject to evolution and change as requirements change, hitherto unknown challenges are discovered, and unforeseen changes of circumstances arise with respect to resourcing, scheduling, tooling or functional constraints. These changes will be captured in the agile process model where possible, and made evident to the client in a prompt and transparent manner.

2   Project organisation

2.1   Process model

For the purposes of this project, an agile project management methodology has been chosen. Several key aspects of the SFM project align closely with the strengths of the agile process model, including:

- Rapid development cycle favouring incremental changes and delivery

- Frequent customer involvement and continuous feedback on developed features

- Small, focused team

- Focus on both functional and automated testing to validate project outcomes

Key details in this section regarding the agile approach and its implementation are informed by [5].

2.1.1   Agile approach

The general philosophy of the agile project management approach requires work to be divided into work items achievable by a single team member within a specified work period, called a sprint. Sprints are typically short (for the purposes of this

project, one week in duration) and generally aim to deliver a functional system in some form.

Work items are added to a backlog and prioritised in order of importance and/or urgency. The expected effort required to completed each item is estimated by the team. If an item cannot be estimated, it is underspecified, and if it is estimated to take longer than one sprint, it must be broken down into smaller items. At the start of each sprint, work items (also sometimes referred to as "stories") are added to the sprint in priority order until the team's capacity is reached for the sprint period, and the sprint begins. Team members work on assigned items until the sprint is complete.

Work items can be classified as subsets of major features or "epics", and a feature can be considered to be complete when all associated work items are completed. This allows a line to be drawn between work items and deliverable features, and aids in estimation of remaining time required to complete a feature (see Section 2.1.4). Sprints are organised into releases, usually corresponding to a major milestone delivery or demonstration (see Figure 1). The release process for this project is explained in detail in Section 5.1.3.

The backlog is an evolving list of items to be completed, which may grow as work is discovered during the development process, and shrink as items are completed.Any team member can create work items, but only the Product Owner may priorities them, and only the Scrum Master may add or remove items from a sprint based on priority. These roles are explained in more detail below.
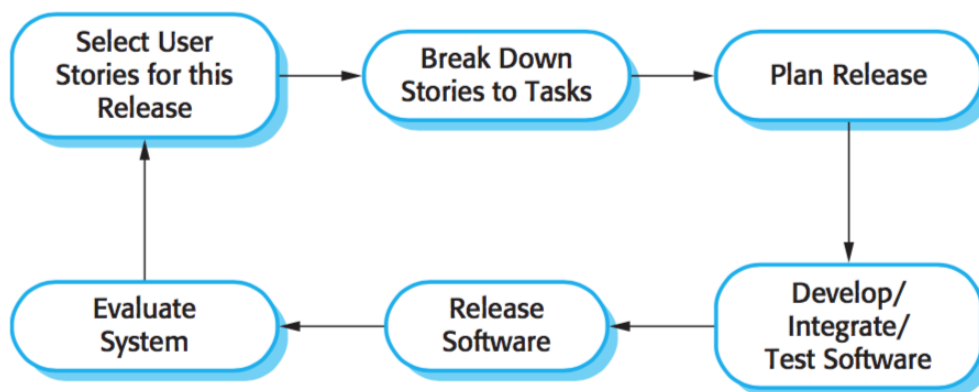

Figure 1: Overview of the agile release cycle [5]

2.1.2   Roles

Apart from the development team itself, the following roles are defined in the context of agile project management:

Product Owner: While a member of the team, the product owner represents the customer's interests. In this capacity, it is the role of the product owner to determine the priority of work items and ensure they are suitably specified and in line with the project requirements as detailed in the SRS.

Scrum Master: The scrum master's role is to ensure the team's workload is correctly distributed, the team is free from external distractions, and informs the product owner if the backlog is not organised or specified to the team's satisfaction.

Strictly speaking, the product owner and scrum master should not be the same person, and both should be separate from the development team where possible. In very small projects, however, such separation may not be possible due to resourcing constraints. For the purpose of this project, the product owner and scrum master roles are fulfilled by the same person under the umbrella term project manager.

2.1.3   Phases

While agile development is fundamentally incremental, it can be beneficial to divide the project into phases to reflect the varying process requirements at different stages of the project. The phases defined for this project are as follows:

Requirements Elicitation: Several meetings are held with the client in order to flesh out requirements and clarify any ambiguities in the original broad specification. It is possible that no formal sprints are conducted during this time, but high-level software architecture is designed and preliminary hardware tests are performed to determine project constraints.

Rapid Prototyping: Formal sprints commence in order to deliver an early prototype of user-facing functionality. This phase serves to give an early concrete form to the outcomes of the requirements elicitation and give opportunity for client feedback prior to the main development phase. This prototype represents the first informal release of software for the project.

Product Development: Based on the outcomes of the requirements elicitation and rapid prototyping phases, formal development of the main software product commences. Releases are scheduled to coincide with deliverable milestones, and sprints are scheduled within release periods.

Post-release: After the release of the final version of the software, there may be a further requirement to finalise as-built documentation and/or testing reports. Formal sprints may not be necessary in this phase.

Note that as per the agile project management approach, there is no standalone testing phase, as comprehensive feature and regression testing is to occur with each incremental release. The specific start and finish dates for these phases are detailed in Section 4.1.

2.1.4   Risks and Reporting

The agile project management approach is not without risk; unlike in a more traditional waterfall-based approach, there is no intrinsic method for tracing a the schedule's critical path. However, various reporting methods are available to project managers in order to satisfy the need to report progress to the client. The primary reporting metric used in this project will be burndown charts (see 2), which report remaining effort required for completion of all work items on a per-sprint and per-release basis.

Estimation represents a significant challenge in the agile process, and therefore requires careful consideration. Often, work item effort is estimated in "points" rather hours in order to ensure that the estimate is a measure of complexity or difficulty of the task rather than the time it will require. This is because different tasks may require different amounts of time to complete the same item, and using points as an abstract measure helps normalise this discrepancy across the team in each sprint [1]. Sprint and/or release progress can be measured using one of two metrics: work items completed or story points completed, depending on the level of granularity required.

Figure 2 shows an example of a burndown chart.

2.2   Roles and responsibilities

In order to facilitate working in parallel, the software development work required to complete the project has been divided into three main functional areas: movement (controlling the motion of the robot in response to operator commands and sensor input, as well as estimating robot position in real-time), vision (gathering and synthesising sensor input for use by the movement and mapping systems), and mapping (constructing a map of the survey in real-time based on sensor input and

Figure 2: An annotated example of a typical burndown chart [6]

estimated position). The breakdown of the team in functional areas, as well as project management roles, is detailed below:

Project Manager: Yann Frizenschaf
> The role of the project manager is to oversee scheduling and resource allocation, as well as to perform the Product Owner and Scrum Master roles as detailed in Section 2.1.2.

Assistant Project Manager: Tao Zhang
> The role of the assistant project manager is to support the project manager in facilitating scheduling and work delegation, and assume the project manager's responsibilities if the project manager is for any reason unable to do so.

QA Manager: Lili Wu
> The quality assurance manager is responsible for ensuring test artefacts are developed and testing activities proceed as per the quality assurance plan (see Section 5.3).

Documentation Manager: Navdeep Singh
> The documentation manager is to ensure that the project documentation is delivered on time and to the required quality standard, coordinating team members' contributions to the documentation, and making sure versioning of

documents is correct.

2.2.1   Functional teams

Movement: Tao Zhang, Yi Lin

Sensors: Lili Wu, Navdeep Singh, Liang Yuan

Mapping: Zeqi Fu, Yann Frizenschaf

3   Risk Management Plan

Risks to successful and timely completion of the project are detailed in Table 2, including the severity and likelihood of each risk scenario. In Table 3, strategies for mitigating the identified risks are detailed. The common identifier across the two tables is the risk ID.

There are six main risk categories. These are technology risks, human resources risks, organisational risks, tooling risks, requirements risks, and estimation risks. In figure 4.2, R01 to R03 are technology risks, R04 to R05 are human resource risks, R06 to R07 are organisational risks, R08 to R10 are tooling risks, R11 to R12 are requirements risks, and R13 is an estimation risk.

4   Work plan

The work plan detailed below represents the preliminary plan for the scheduling, and execution of the project works.

4.1   Work Activities

The start and finish dates for the project phases introduced in section 2.1.3 are detailed in Table 4.

4.2   Milestones

The delivery dates for project artefacts and key milestones are detailed in Table 5.

Risk ID	Description	Severity	Likelihood
R01	Software defects cause test failure	Medium	High
R02	Team members leave the team	Medium	Low
R03	Data loss	High	Medium
R04	Underestimation of required effort	Medium	Medium
R05	Communication difficulties between team members	Medium	Low
R06	Tasks are not allocated reasonably	Medium	Medium
R07	Unexpected changes to requirements	Medium	High
R08	Hardware damage to robot	High	Medium
R09	Failure of robot remote communication	High	High
R10	Loss of robot hardware	High	Medium
R11	Schedule slippage	High	Medium
R12	Adding new requirements	Medium	Medium
R13	Estimated testing time not sufficient to complete test activities	High	High
R13	Wireless reliability and data transfer	High	Medium
R13	Different Operate system and Software portability	High	Medium

Table 2: Identified risks

Risk ID	Risk Cause	Risk mitigation plan
R01	Insufficient testing	Implement comprehensive test plan
R02	Members withdraw from the course	Implement role-based redundancy
R03	Computer hardware failure	Distributed backups
R04	Differences in skills and experience across team	Arrange for team members to complete tasks that match up with their individual skills
R05	Team members' native languages are different	Important decisions confirmed in written form
R06	The working plan is not suitable for an individual	Ensure flexibility of task assignment as per agile process model
R07	Client requirements change or work is discovered	Perform regular requirements review
R08	Inadequate safety measures during testing	Ensure safety measures are reflected in functional test plan
R09	Primary wireless communication method is unstable	Ensure backup communication plan is tested and software fail-safes are implemented
R10	Insecure storage	Ensure robot is stored in secure locker when not in use
R11	Inadequate estimation process	Adapt estimation process as per agile process model
R12	Discovered work, unexpected bugs, new requirements	Negotiate feature and schedule change with client at earliest opportunity
R13	Schedule slippage	Prioritise testing of existing features over new feature addition

Table 3: Risk causes and mitigation

ID	Project phase	Start date	Finish date
1	Requirements elicitation	09/08/2016	22/08/2016
2	Rapid prototyping:	22/08/2016	13/09/2016
3	Product development	13/09/2016	30/10/2016
4	Post-release	30/10/2016	01/11/2016

Table 4: Schedule of major project phases

Deliverable/Milestone	Due Date
Software Requirements Specification	01/11/2016
Software Project Management Plan	01/11/2016
Software Design Document	01/11/2016
Testing Report	01/11/2016
User Manual	01/11/2016
Milestone Software Demonstration 1	13/09/2016
Milestone Software Demonstration 2	04/10/2016
As-built Software Demonstration	01/11/2016

Table 5: Delivery dates for project artefacts and milestones

4.3   Schedule and resource allocation

The estimated workload for the team as a whole is established once for each phase of the project, at the beginning of the phase. Additionally, adjustments in the schedule will be made for each of the team members on a per-sprint basis, depending on the workload for the given period of time. The goal for each sprint is to complete the allocated work as a team, so tasks can be reassigned to team members who finish their assigned tasks early.

In general, software development works will be divided by functional area as detailed in Section 2.2, and documentation work will be divided by document section. However, in some cases team members may be required to assist colleagues outside of their functional team, as per the agile approach, in order to achieve sprint/release completion.

In some cases, it may be necessary to push unfinished tasks to a later sprint if the required effort was underestimated. Similarly, it may be possible to bring tasks forward if the total sprint effort was initially overestimated. In this way, the inherent error in the agile estimation process is intended to even out [5]. If effort is found to be consistently under-estimated, however, the team will be required to re-calibrate the estimation process by intentionally adding some overhead to estimates going forward.

5   Supporting plans

5.1   Configuration management

Configuration management serves to ensure that the current design and build state of the project is known. It allows access to a historical record of project state relevant for both project management and development activities. The benefits of a comprehensive configuration management plan include[4]:

- Maintaining the integrity of the project.

- Tracking the changes during development.

- Providing an ability to trace the process that from requirement to product.

- Reducing the cost of maintaining project.

- Providing a stable environment for software development.

5.1.1   Software configuration management plan

The architecture of the Prospector software project is broken down into hierarchically-arranged functional areas as listed in Table 6. Each of these functional areas is implemented as a Java package in the project structure, and may be the child of a parent package, or a root package. In this way, functional separation of concerns is achieved and parallel development is enabled; each functional area can be treated as a black box by other functional areas, provided appropriate interfaces are defined.

5.1.2   Version control

Version control during all project phases is supported by the Git VCS (version control system). While Git is not a complete configuration management solution in and of itself, it enables tracking of software development history and tagging of snapshots in the development cycle, which complements the release management plan detailed in Section 5.1.3.

5.1.3   Software releases

Each software release represents a working version of the software system which is functional and feature-complete with respect to the requirements of the correspond-

Functional area	Description	Package	Parent package
Prospector	Root package	prospector	-
Movement	Motor control, position estimation	movement	prospector
Vision	Sensor input synthesis	vision	prospector
Mapping	Map generation	mapping	prospector
Operations GUI	Operator HMI	opsgui	prospector
Data IO	Database and file management	dataio	prospector
File operations	Map file saving and loading	fileops	dataio
Database operations	Map database management	databaseops	dataio
Unit tests	Automated software testing	test	prospectortest

Table 6: Software architecture configuration

ing milestone. Each release will be accompanied by suitable set of release notes detailing the relevant major features and/or bug fixes pertinent to the release. The software itself will be provided as a pre-built executable file (or files), as well as an archive of the source code and instructions for build the software from the source using the specified build tools (ANT and MAKE).

Releases are created by tagging specific revisions in the Git VCS repository, meaning a release can be uniquely identified as a snapshot in the development history of the software.

In general, the Git repository will follow a common branching model expressed eloquently by [2]. There are two main branches, called *master* and *develop*. All ongoing/incremental development work will be performed on *develop*, and only when the software on that branch is ready for a release will all its revisions be merged to *master*. The revision which is merged will be tagged as a release and built against the automated test suite (see Section 5.3). This branching model is shown in Figure 3.

5.2   Documentation plan

There are a number of documents that will be produced during the developing process of our project, and some are listed below. Note that the requirements for the content of these documents are discussed in Section 1.4.
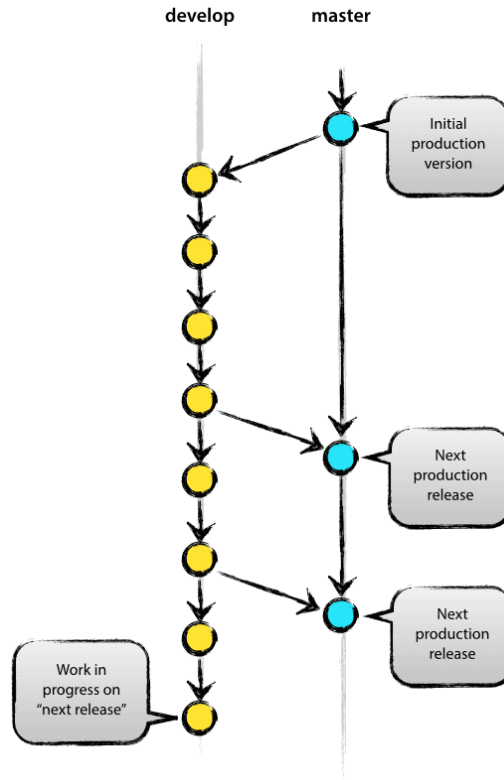
Figure 3: The proposed branching model [2]

1. Software Requirements Specification

2. Software Project Management Plan

3. Software Design Document

4. Testing Report

5. User Manual

5.2.1 Documentation preparation and versioning

Each document has a unique name, and a version number to be updated with each official revision/release (beginning with 0.1 for initial draft releases). While documents are under development prior to and between releases to the client, modifications and individual contributions will be tracked via the Git VCS.

Item	Standards	Participants	**activity**
SRS	SRS checklist	Clients and All team member	Requirement Review, Management Review
SPMP	SPMP checklist	All team member	Requirement Review, Management Review
Source Code	Interface code file	Developers	Code Review, Management Review
SDD	SDD checklist	All team member	Code Review, Test Review
Test report	Testing plan	Testers	Code Review, Test Review
User manual	GUI design	All team member	Requirement Review, Management Review

5.3   Quality assurance plan

Quality assurance has very important role in the Prospector SFM project. The main goals of the quality assurance plan are as follows:

- Provide the ability to quantitatively verify that the project satisfies the elicited client requirements

- Ensure that the delivered software meets a reasonable level of quality

- Provide the ability find and remove defects during the development phase of the project

- Ensure that the delivered documents accurately describe the delivered software and relevant processes

5.3.1   Software quality

Software quality will be assured in the following ways:

Unit tests: Fine-grained automated tests which are run as part of the software build process. Each self-contained, nontrivial software function which returns a value or alters internal state should have an accompanying unit test to ensure the output or altered state matches the provided input. Software with failing unit tests may not be released, and should not be checked in to the VCS where avoidable. Unit testing is the responsibility of each team member when developing self-contained software classes and methods.

Functional tests: High-level tests which demonstrate user-facing features of the software in order to quantifiable prove that the associated requirements have been met and the software is suitably defect-free. Developed by the project team under the guidance of the QA manager

System tests: After building different part of the system, we have to make sure all the components can work well together. Thus, system tests are trying to make sure all the core functions can work together which are required by client.

Pressure tests: The system is required working in a long time that we have to do the pressure test. Pressure tests can show more software bugs even hardware defect. Not only running robot in along time by also in some rigor conditions.

5.3.2   Activities

Requirements review: This bulk of this activity takes place in the requirements elicitation phase of the project (see Section 2.1.3), but becomes an ongoing background process thereafter to ensure that development work is not proceeding tangentially to the core project requirements. This activity will be conducted by the entire project team under the guidance of the project manager.

Management review: This is an ongoing activity to be conducted by the project manager. An overall view of the schedule should be maintained at least at the end of each sprint. In the case of schedule slippage, and modifications to scheduling or requirements should be negotiated with the client by the project manager on behalf of the team. The purpose of this activity is to ensure the client's expectations are in line with the likely project outcomes and delivery dates, and schedule slippage can be minimised where possible.

Code review: Code reviews will be conducted within functional teams on a per-sprint basis. At the end of each sprint, each team member who has contributed code changes to the software repository must ensure that the changes have been reviewed by another member of their functional team, and any feedback has been incorporated. In case of a dispute over coding style or content, a third member of the project team should serve as the tie-breaker.

Document review: Each document will be reviewed at least once by every contributing member of the team prior to client release. Client feedback on draft releases will be incorporated by a relevant team member, and the changes will be reviewed by at least one other team member for completeness and correctness prior to re-release.

Test review: All functional tests should be reviewed by both the QA manager and the functional team relevant to the functional area under test. Where possible, the high-level requirements for the outcomes of functional tests should be reviewed by a client representative.

Test review: All functional tests should be reviewed by both the QA manager and the functional team relevant to the functional area under test. Where possible, the high-level requirements for the outcomes of functional tests should be reviewed by a client representative.

6   Glossary

GUI Graphical User Interface

HMI Human-Machine Interface

LeJOS The Lego Java Operating System

SFM Sea Floor Mapping

SRS Software Requirements Specification

SPMP Software Project Management Plan

SDD Software Design Document

VCS Version Control System

7   References

[1] Cohn, M 2014. *The Main Benefit of Story Points*, viewed 6 September 2016, <https://www.mountaingoatsoftware.com/blog/the-main-benefit-of-story-points>

[2] Driessen, V 2010. *A successful Git branching model*, viewed 6 September 2016, <http://nvie.com/posts/a-successful-git-branching-model/>

[3] Milanese, D and Weerasinghe, A 2016. *Software Engineering and Project - SeaFaults Mapping Robot.* Project specification version 1.0

[4] Prospector Team 2016. *Software Requirements Specification: Prospector Seafloor Mapping System.* Version 0.1 (Draft).

[5] Sommerville, I 2011. *Software Engineering*, 9th Edition, Addison-Wesley, Boston

[6] Rasmusson, J 2016. *Agile in a Nutshell*, viewed 5 September 2016, <http://www.agilenutshell.com/burndown>