Methods and Plans

Our approach to this software development project will be to employ a combination of two methodologies. First of all we will be using the agile method **SCRUM**, in combination with concepts of test driven development [1]. There are numerous advantages of using agile development (specifically using the SCRUM method) and test-driven development, however both are not without their drawbacks. [2]

The main reason we chose an agile method is for the ease it allows us to adapt to changes in our requirements. As software is delivered regularly after 'sprints' [3] (set lengths of time where software is developed) comments and drawbacks can be noted and new requirements can be added to the log (a document contain new features, requirements, and problems) during the next sprint. This is incredibly important for the later assessments where requirements will be changed, and we will be working with existing codebases that are not our own.

Risks can also be precluded before they become an issue by having regular meetings and discussions within the team. Once any risks have been mitigated there may be additional decisions on the project which can then be added to the log of features and requirements, and so implement them when necessary. This ties into the innovation aspect of software engineering as well. If a member of the team identifies a key feature that may have been missed or needs improving, it can be added onto the log. The same process can be applied to issues or bugs.

Agile development does also come with some drawbacks, however. Due to agile development's nature of adapting to change and its style of iterative development, there is a large possibility of "scope creep" [4]. This can be constituted by the team leader not taking enough of an active approach with programming, and requesting features that are unattainable. It can be corrected by imposing strict deadlines and carefully managing the team however it is a potential issue that may need addressing. It is also difficult to accurately plan throughout the entire project as tasks or requirements may be added to the log due to the ongoing nature of the project. This may mean that a single task may take up to several sprints to accomplish, lengthening the development time.

As mentioned before we will also be utilising test driven development within our sprints. Due to test driven developments innate ability to speed up development, development of features during a sprint will be increased. We can then receive feedback and build on that. A further benefit is that due to the very nature of test driven development, ensuring features are tested does not need to be added to the log of tasks. Therefore the team can solely focus on implementing new features, and improving the features we already have developed [5].

In practice this system functioned well[1]. By utilising test driven development within our sprints we were able to implement testing as we were progressing. In addition by utilising a shared team log, the team could all see how the current sprint was progressing and what needed to be done.

Tools and Usage[2]

---

[1] **Added** discussion of how approaches worked in practice

[2] **Moved** *Lucidchart* discussion to their relevant sections to streamline document.

During our software development we will be using a variety of tools that will assist us in both developing and collaborating on our project. The first tool that we will be using ***Facebook Messenger*** as our main form of communication. The application's useful style of instant messaging will allow us to quickly communicate to the entire team.

Another tool we will be using for planning is ***Trello***. This is a tool that will allow us to view the progress on our project in real-time, both on desktop and mobile. It uses boards to manage groups of assignments (e.g. one list is entitled *requirements*, containing all the stages in creating the requirements section of the assessment) and within these boards are cards (using the requirements metaphor again, there would be a card entitled "first draft" etc.). These include dates for when the card (task) is due, and a checklist for whether it is done. We are using this tool to track and manage our development.

A tool that we will be using is ***Google Drive***, which we will be using as a platform to share documents. First of all it allows us as a team to quickly collaborate on one document, and it is easy to create a file structure we can all use; creating a central hub where all members of the team can access documents relevant to their section of work (and others, for helping, proofreading etc.). It allows members of the team to access the documents from different device making continuation of work easy. It is also useful to use as a central repository so if a team-mate is ill, we can still access and update their work if it is stored on the Drive. Finally the Drive supports real time updating so all team members see the most up to date versions of documents.

A similar tool that we will be using is ***GitHub***. We will be using as our version control tool to keep our codebase up to date, and will be most useful for the implementation sections of the assessment. It is widely used, so there will be a wealth of information on how it is used from both fellow users and standard documentation. We also found that many members of the team are also familiar with GitHub and its use. As both tools (GitHub, Drive) are used as a repository for the team's documents, we will be managing them in different and strict fashion, keeping the two as separate as is feasible. The drive will be used for sharing documentation, resources and notes, while our GitHub will be used for sharing and merging the actual code. This is due to GitHub's easier facility to share code and the Drives easier nature to share documents and access them from multiple devices.[3]

The main API that we will be using is ***LibGDX***. LibGDX is a library that assists with the creation of cross platform game development. We decided to use this tool for a multitude of reasons: one being that it has a wide user base, meaning that there will both be community tools, documentation and support, and due to its popularity it may be a marketable feature during the third assessment part of the project as other teams may have some experience using it in development. In addition its ability to deliver cross platform support is ideal to the scenario as it will allow us to deploy the game to both the computers in the CS building, as well as deploying it to potential students' phones on both iOS and Android platforms. In addition to LibGDX we will be using the IDE IntelliJ [6] .LibGDX supports **IntelliJ** natively which is beneficial when setting up our project, allowing the project to be easily created with the needed dependencies

---

[3] **Added** discussion of *Drive* vs *GitHub*.

and file structure. Furthermore members of our team have previously used IntelliJ before, allowing them to share their experience with the team.[4]

To assist in asset development, we will be using the **tool tiled map editor** which will be used to create maps easily and test them instantly, rather than having to create them externally and then integrate them. This tool also comes with native support for LibGDX, and has intuitive support for effects such as animated tiles, image filters and translucent tiles. In addition due to the nature of tiled, we can easily swap tile sets allowing us to prototype with "Programmer art" and then replace with a final tile set without having to redraw the map [7]. [5]

Team organisation and justification:

Our team is organised into a hierarchical structure. There will be one Team Leader that is responsible for the overall management of the team. This team leader will also take the position of 'Scrum master' (the SCRUM development term for the leader in terms of how information is exchanged). Below the team leader the hierarchy will split into two substructures. These substructures are titled *Meeting Roles* and *Project Development*. Directly under the management of the team leader under the project development side is the *lead software developer* and the *report editor*. These will be directly responsible for the roles below them. The number of software developers and documentation creators are fluid, meaning that members of the team can be assigned new roles to keep up with the demands of development but still know the command structure. This structure is represented to the left and again in the appendix (Item 3.1).

We decided on having one overall team leader as firstly, once agreed on a direction, they can keep us focused to it and stop the team becoming distracted. Plus with one leader providing direction members of the team can focus on their strengths, rather than having to make managerial decisions in areas they may be weak in. Also due to the hierarchical structure all members of the team know who should be working with who and which members of the team are responsible for each section of development. One team leader can also provide a morale boost in certain situations, such as "crunch time" (the last few days of development, which is particularly stressful).

That is not to say that there are no drawbacks to one leader. First of all if the team leader fails in their job then the rest of the structure collapses meaning that the team may need to be restructured costing time, or a new leader introduced; potentially causing friction within the team. To link with this is the inevitable clash of personalities, leading to division within the team, and the potential for a hostile working environment. However another approach to the situation is if that there are any problems, blame can fall solely on the team leader rather than take the responsibility as a team.

The scrum development team will also be headed by the lead software developer, whose responsibility is to ensure that each iteration of the product is delivered properly and to organise each sprint correctly. This team is also autonomous and should be able to function without direct input. This is the same as the report team, which should be able to act under the report editor without needing constant management.

---

[4] **Added** IntelliJ discussion as this is beginning to be used for this assessment.

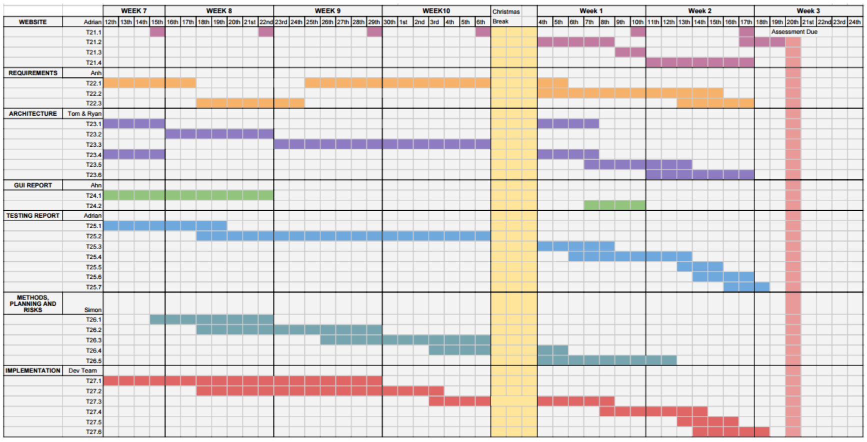
[5] **Added** Tiled Editor discussion as this is also now been introduced.

Though assessment 2 splits the team into two sections, this is not to say that the teams organisation may not change somewhat. What this means is that members of the team should have some familiarity with all sections of development. This means if necessary that the team leader can change teams to cope with the current developmental needs, an example of this could be towards the end of the development, where the majority of the development team may be moved to the report team to assist in document creation rather than implementing unnecessary features. This links to our risk assessment as well if our code isn't created with enough time to fully finish the documentation creation with enough polish. [6]

Systematic Project Plan[7]:

To outline our plan we will be mainly using a Gantt chart. In creating the Gantt chart we broke down the tasks for each assessment into their sections then as a group we broke these down into their core components following the principles of top down design. From there we analysed these components and mapped these into a basic shape taking into account which tasks would need to be done before others. Once this version was created, we updated it to a more precise schedule including dates such as team meetings, stakeholder meetings and due dates.

The Gantt chart above shows how it can be used to display the start date and end date of a piece of documentation. This is displayed by the bars of colour running horizontally with where the bar ends when that task is due. The Gantt chart can also intuitively show how tasks are dependent on one another. To use an example from assessment two, I refer you to the diagram. To begin the task T25.5, tasks T25.1-T25.3 must be finished. Shown by the fact that T25.1 does not start till the relevant tasks are finished is finished.



To explain the screenshot above, the Gantt chart intrinsically shows how it can be used to display the start and end date of a task. This is displayed by the bars of colour running horizontally with where the bar ends when that documentation is due (the piece of documentation due can be easily found by the associated key.). The Gantt chart is also important for determining the critical path of the project. By utilising task dependencies we can calculate how long a project should last (according to the plan). Though this may change once the plan has been brought into reality, it can give a good idea of how long the project should run.

---

[6] **Added** section relating to possible changes in team organisation

[7] We have included a small clipping of the first Gantt chart in this document however the full Gantt charts for the project are available in sections 3.6-3.9 of the Appendix.

Bibliography

[1] Agiledata.org, "Introduction to Test Driven Development (TDD)", 2016. [Online]. Available: http://www.agiledata.org/essays/tdd.html. [Accessed: 12- Jan- 2016].

[2]T. Stober and U. Hansmann, *Agile software development*. Heidelberg: Springer, 2010.

[3]Ken Schwaber, Jeff Sutherland, *The Definitive Guide to Scrum: The Rules of the Game,* http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf

[4]TeamGantt, 'Scope Creep – The Two Dirtiest Words in Project Management', 2014. [Online]. Available: http://teamgantt.com/blog/scope-creep-the-two-dirtiest-words-in-project-management/. [Accessed: 09- Nov- 2015].

[5] L. Madeyski, *Test-driven development*. Heidelberg: Springer-Verlag, 2010.

[6]JetBrains.IntelliJ. https://www.jetbrains.com/idea/

[7]Tiled. Thorbjørn Lindeijer. http://www.mapeditor.org/download.html