Approaches to Software Engineering:

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. There are numerous advantages of using agile development (specifically using the SCRUM method) and test-driven development, however both are not without their drawbacks.

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' [1] (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" [2]. 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. ***Scope creep* also constitutes the changing of the requirements mid-development. This is also important to be managed properly, by keeping the code well documented and easy to maintain from the implementation side. The deadlines for requirement changes will also be strict, so sprints managed properly.** *This was added to our report to take into account that due to the upcoming requirement changes for the next assessment, it is very relevant that this is taken into consideration.*

Our team will be utilising agile testing to test the product which will be discussed in more detail on the testing document. **The previous team had been testing their code in *JUnit* but we have decided to continue with our method of testing, as our team have little experience with *JUnit* and it was decided that with the short assessment time frames, we should stick with our current testing regime. It could be an issue if we didn't test correctly then we couldn't guarantee a complete product.**

Tools and Usage:

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 teammate 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.

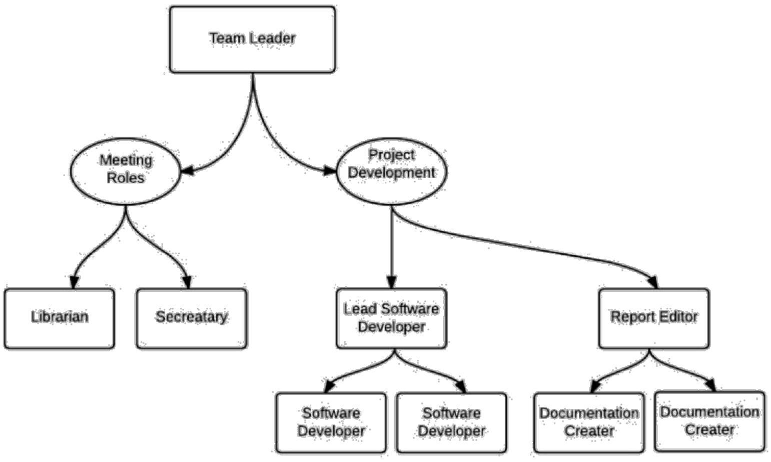
A similar tool that we will be using is *GitHub*. This is what we will be using as our version control tool to keep our codebase up to date, and will be most useful for the implementation sections of the assessment. It is widely used, so there will be a wealth of information on how it is used from both fellow users and standard documentation. We also found that many members of the team are also familiar with GitHub and its use.

Another tool that we will be using is *Lucidchart*. This tool will be used for designing our architecture in UML. First of all Lucidchart is the most accessible, both being web based and therefore working on all operating systems it was also very easy to learn how to use. Lucidchart also had team collaboration support; it integrates with Google Drive / Documents (which we are currently using as our main file sharing system. For these reasons we concluded that this would be the best choice for our UML tool.

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. It also includes native tools to assist in asset development as well, one such example is the tile 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.

The team we selected in our swap process was already using GitHub, Google Drive and LibGDX and tools in their team's project which made the process of adopting files and methods straightforward. We also ensured that files for assessments were kept separate to ensure they didn't get 'mixed up'.

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 (Adrian W) 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* (Huw T) and the *report editor* (Ryan E). 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.

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.

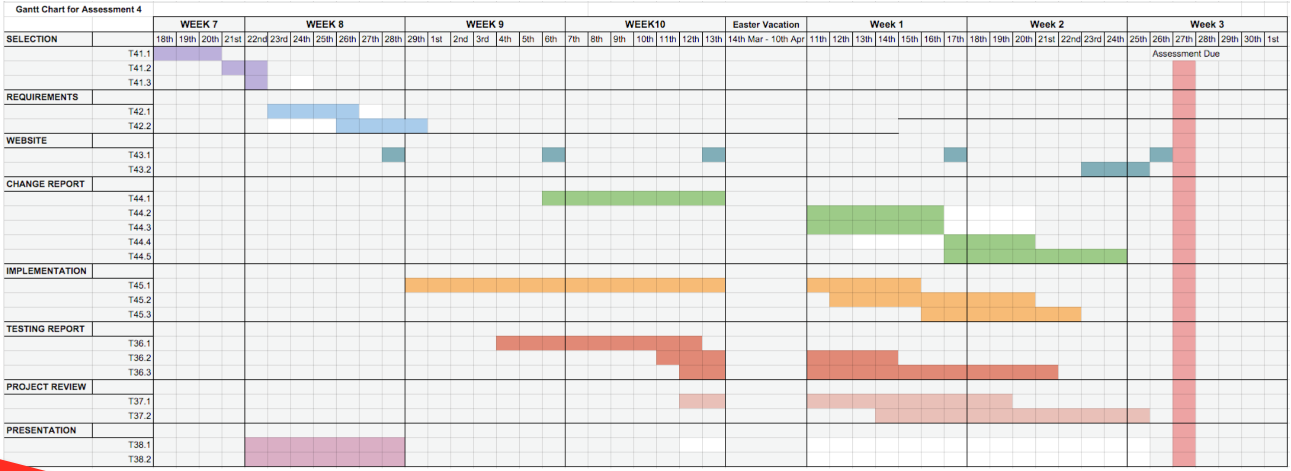
The scrum development team will also be headed by the lead software developer, whose responsibility is to ensure that each iteration of the product is delivered properly and to organise each sprint correctly. This team is also autonomous and should be able to function without direct input. This is the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

None of this has been modified since the last assessment as we consider our team structure to be appropriate; the Team Leader will continue to manage the change during the swap process, with the assistance of the lead software developer.

Systematic Project Plan:

To outline our plan we will be mainly using a Gantt chart. In creating the Gantt chart firstly 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 below 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. Systematically for Assessment 4, an example is that Implementation task T45.1 carrying out the requirement changes to the code cannot occur until the requirements have been analysed and the team has discussed their implementation.



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 can also intuitively show how tasks are dependent on one another. To use an example from assessment four again, I refer you to the diagram. To begin the task *T44.4*, tasks *T44.2* and *T44.3* must be finished. This is shown by the start dates and the fact they do not run concurrently.

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. To use an example from the above diagram you can see that the critical path for the change report documentation to be done is around three weeks.

We have included a small clipping of the Gantt chart in this document however the full Gantt chart for Assessment 4 are available in the Appendix.

Modified here: we have changed the description of the Gantt chart to use relevant examples from the following assessment and replaced the diagram.

Allocation of Assessment Tasks
Note that some sections of the assessment are large, time consuming or are deemed appropriate to have two or more people working on them; thus people working on tasks are not mutually exclusive.

Team Member	Assessment Section	Tasks	Explanation
Adrian	Website, Testing Report	T41.1-3, T43.1-2, T36.1-3	Will ultimately make and process the change decision and continue to monitor the testing report (cont. from Assessment 3)
Anh	Change Report	T44.1,3,5	Responsible for the bulk of the change report, based on experience from previous assessments.
Huw	Change Report, Implementation	T44.4, T45.1-3	Responsible for the implementation development side - has experience in development and has done so previously.
Ryan	Change Report, Implementation	T44.4, T45.1-3	Responsible for the implementation development side - has experience in development and has done so previously.
Simon	Project Review Report, Presentation	T47.1-2, T48.1-2	Responsible for planning the presentation using experience, and the review report
Tom	Project Review Report, Change Report	T44.2, T47.1-2	Responsible for the review report and for the architecture documentation in the change report (as in previous assessments)

References:

1. Ken Schwaber, Jeff Sutherland, *The Definitive Guide to Scrum: The Rules of the Game*, 2014. [E-Book] Available: http://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf
2. 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].