Method Selection and Planning

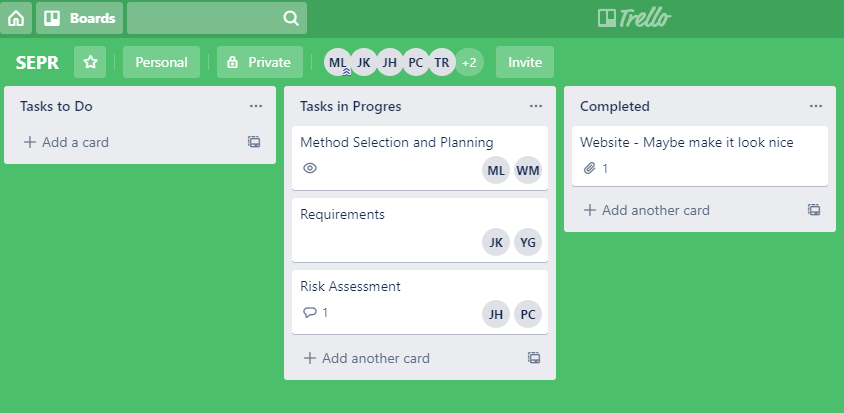
Give an outline and justification of the team’s proposed software engineering

methods, and identify any development or collaboration tools that the team plans to

use to support the project or the team working (10 marks, \_ 2 pages).

* Scrum agile sprints etc

The software engineering methodology that we chose was Scrum. Scrum is an agile methodology allowing smaller teams to work efficiently to strict deadlines. This means that the focus is on iterating smaller sections of the whole project rather than trying to build the entire thing in one go. The advantage of this is that any short notice requirement changes are easier to deal with and so means the team is more flexible. We will use discord as our communication platform for our scrum meetings which we will hold daily (🡨**MAYBE CHANGE THIS LATER).** In these meetings each member takes a couple minutes to explain what they are doing on the day and how other members tasks may coincide with their own. Discord is the obvious choice for our team as we live all over York, we can communicate verbally at any time and this allows explanations of ideas to take much less time than text-based messaging. Further to this we will use Trello to track tasks, whether that be tasks to do, tasks in progress, or completed tasks, see example below.



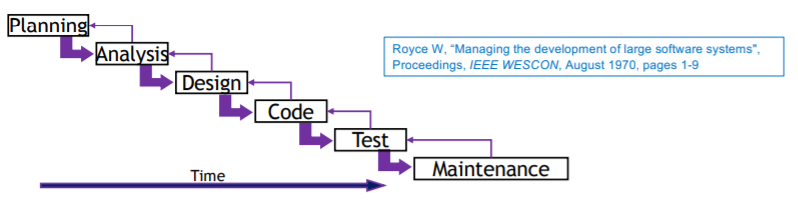
Trello allows us to comment and assign members to each task allowing us to see at a glance what needs doing and who could potentially do it. This meshes well with the agile methodology as it allows tasks to be quickly edited and swapped out. We will also not hesitate to contact the customer if any design issue or ambiguity arises, this is another core feature of an agile methodology as embedding the customer in the team allows us to better understand their needs.

* Research classical dev methodologies and explain why we didn’t use those

We chose against classical and plan driven methods as these wouldn’t allow us the flexibility of being able to change requirements. One such classical methodology is the Waterfall model, this is a very linear model where all requirements are drawn up first, then the software architecture is designed and then all the software is implemented. This linear model does not offer much scope for change and as such does not suit our project as we anticipate change to the requirements that exceeds 1% change per month. Further to this, plan driven methods focus heavily on documentation and so tend to weigh the project down with unnecessary bureaucracy. This may well work for scenarios such as the client being remote, or where extensive documentation would be needed upon release anyway, but we thought that neither of these apply to this project. See below figure 2 on the waterfall methodology.

Due to the structure of the project we will roughly follow the RUP (Rational Unified Process) anyway but we aim to have smaller gaps between each stage. For example, as I am writing this the requirements are also being written up which is one of the first parts of the project that would be heavily focused by RUP. We may, as a part of this, attempt to construct stories to help us anticipate potential issues that may arise in future. For example we could construct one of the typical user to figure out what features we need to make the product as intuitive as possible.

On top of Discord and Trello we will also use GitHub, this is the obvious choice for any software development project as it allows easy collaboration and has many 3rd party applications that can push/pull etc. This means each team member can use their favourite application and it will have no adverse effect on the other members of the team.



As we are coding this project in Java we are following an object oriented design, this has many benefits for our team. Firstly, code is much easier to compartmentalise, we can get each member to write specific classes given a specification and they should all mesh together in the final product. Secondly, object-oriented design allows us to abstract out key design features into objects, for example a fire engine is its own object with its own methods and attributes, and this makes designing the overall product more manageable and more easily translatable to code. Further to this we can use inheritance to reduce the file size of the game and make designing certain aspects easier. Following our fire engine example, this object could inherit from a class called “moveable” which is an item in the game capable of moving and therefore we only need to write the moving code once.

In order to design and prototype the architecture we are going to use UML (Unified Modeling Language). This will allow us to graphically represent all of the classes and the relationships between them enabling us to refine the architecture well enough for us to implement. It will also enable us to bring this visual representation of our architecture to meetings with the client if they so wish. Further to this a visual representation of our system will be easier to understand than a written one meaning we spend less time re-reading documentation and more time designing and implementing.

Outline the team’s approach to team organisation, and explain why the chosen

approach is appropriate for both the team and the project (5 marks, \_ 1 page).

* No distinct leader, all members contribute meaning we take advantage of everyone’s skills
* Discussions had on what needs doing and who prefers what, attempting to fit the right tasks to the right people for a higher quality product
* More people assigned to difficult or longer tasks
* After a task is completed, members that worked on it are reassigned to a task with high priority that they are suited to

These approaches ensure that each team member has an equal say and are placed where they will perform best. The tasks with highest priority will be completed first to ensure that tasks with dependencies aren’t delayed and that the core features will be implemented. Reassignment to tasks that are taking time will prevent bottlenecks in development.

Give a systematic plan for the rest of the SEPR project. Your plan should lay out the

key tasks for assessments 2 to 4. You should provide a detailed plan for **the the**

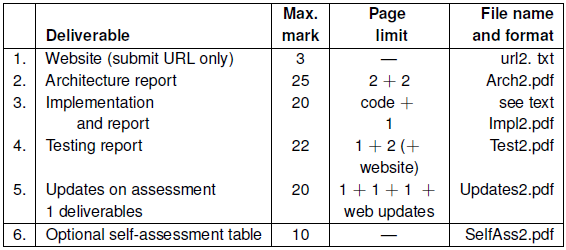
**software engineering tasks required for assessment 2**. Tasks should indicate the

earliest starting date and the latest finishing date, as well as **task priorities**. The plan

should also identify a **critical path and task dependencies** (5 marks, \_ 1 page).

Assessment 2: Preliminary Implementation, due: noon, Spr/3/Mon

For assessment 2 you are required to implement two fire engines and three ET fortresses. You should not implement patrols or the mini game. Also, in this initial version of the game, ETs should not be able to destroy the Fire Station and their fortresses should not improve over time



URL should link to:

* All A2 deliverables and A1 versions
* The executable of the game so far
* Executable test plan and results from testing so far
* User manual – one editable and one .pdf

Implementation:

* Provide documented code for a working implementation of the game, must also include executable JAR of the game
* State any of the features not present

Architecture report:

* Concrete architecture showing code structure
* Justification of concrete architecture

Software testing report:

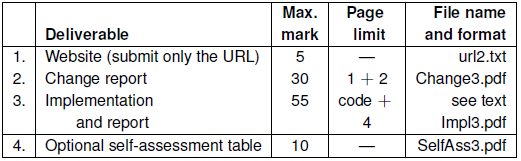
* Summarise testing methods and approaches
* Brief report of actual tests
* Precise URLs of testing material on the site

Additional tasks:

* Updated requirements
* Methods, plans update
* Risk assessment mitigation update

Assessment 3: Selection, Extension and Integration, due: 17/02/20

* Evaluation and selection of a product to develop further, occurring between 20/01/20 - 27/01/20:
  + Presentation of our product: 23/01/20 or 24/01/20
  + The decision will be made based on current quality of product, how well it follows requirements, the difficulty of implementing the remaining features, and quality of documentation.
  + Registering selection made by emailing lecturers
  + High priority, taking no more than a few days so development can begin
* Implementation:
  + Completion of any missing features from assessment 2
  + Features to add:
    - 2 or more new fire engines
    - 3 or more new fortresses
    - Minigame
    - Patrols
    - Fortress strength scaling with time
    - ETs ability to destroy fire stations
  + Documentation of code
* Change report:
  + Formal approaches to change management
  + Testing report
  + Methods and plans used



Assessment 4: Selection, Requirements Change, due: 29/04/20

* Evaluation and selection of a product to develop further, occurring between 17/02/20 - 24/02/20:
  + Presentation of our product: 23/01/20 or 24/01/20
  + Similar evaluation and priority to selection in assessment 3
* Reviewing changes to the brief or requirements of the project and implementing changes to satisfy new requirements
* Implementation of changes to the product
* Documentation of code
* Development of presentation of product to a client
* Assessment 4 due: 29/04/20

