University of York

Department of Computer Science

SEPR - Assessment 2

Updated

Method Selection and Planning

Team Craig

Thomas Burroughs

Huw Christianson

Joseph Frankish

Isaac Lowe

Beatrix Vincze

Suleman Zaki

Method Selection And Planning

Justification of Software Engineering Methodology

Software engineering methodologies are used to provide a framework for developing software within software engineering teams. The framework is used to structure, plan and control each of the individual processes required to produce a suitable end product. The chosen software development method must be justified by achieving the best results for money and time spent [1].

On initial research, it became clear that a plan-driven approach used by large, industry firms was not suitable for this project, as more time is spent on how the system will be developed rather than developing and testing [2]. This is not suitable for this project as deliverables must be produced in four short assignments. After assessing the team's potential organisation and requirements for the customer, it became clear that an incremental agile method would be best suited for this project. This is due to our inexperience, our small team size and the relative short time frame of the project.

Our lack of experience means that creating a plan for the whole project at the beginning, would be unwise and likely not efficient. The flexible, iterative nature of some agile methods, such as Scrum, will allow the team to make revisions on work regularly. This will allow for issues arising from inexperience to be resolved, and allows amendments to plans, as our knowledge grows. Taking this into account, the team will use the Scrum approach, however, alterations will be made to accommodate for the nature of the task and the personal needs of the team. The Scrum method is characterised by the living backlog of prioritised work to be done; the completion of backlog items in a series of sprints (short iterations); a scrum (daily meeting) where progress is explained, upcoming work discussed and any issues raised [1]. Some of these characterisations are not appropriate for this project and have therefore been modified. Short daily meetings have been replaced with two longer weekly meetings to adjust for the team's university schedule. Sprints have been reduced to one week due to the limited time-frame of the assessment, especially assessment 3, and the Scrum master's role has been adjusted to only be responsible for the coordination of the meetings and has been re-defined as the Meeting chair.

The Scrum method enables the whole team to observe everything and consequently improve communication [2]. This is particularly useful for our small team, as each team member will, at some point, provide input on every element of the deliverable, as explained in the team organisation section. If each team member has knowledge of the current state and proposed plan for each element, collaboration will be more organised and the whole project will be more cohesive. This helps to further justify our choice of software engineering methodology.

Further justification for not using a plan-driven approach, such as the waterfall model, is the requirement for large amounts of documentation used for presenting the deliverable to the customer. Throughout our project we are able to receive feedback from the customer in informal face-to-face meetings. This enables the customer to have a greater input on the product, and allows them to make necessary comments on designs and adjustments to requirements. As we have regular face-to-face communication with our customer, taking an agile method is suitable as these methods promote customer involvement in the development process and the minimal production of documentation [2]. Despite this, adjustments to the agile method will be made as documentation is required as part of the assessment, and will be used to minimise misunderstandings and to share knowledge between the team and with the customer.

The assessment document outlines that during the project there will be changes in requirements due to changing customer needs. Agile methods welcome changing requirements, even late in development, to provide a competitive advantage for the customer as outlined by the Agile Manifesto [3]. Using our agile method means we can incorporate changing requirements as design development and requirements engineering can happen at the same time, further justifying our use of the Scrum method.

Development and Collaboration Tools

To maximise productivity and ensure an equal ownership of the project, our collaboration, communication and development tools must be accessible by all team members and future teams who may take over the project. Due to this, each one of our tools is free, easy to learn and most likely familiar amongst students or software engineers.

As a group we discussed a suitable platform which we could all use to communicate daily. A uniform decision was made to use WhatsApp, as each member was already experienced with the app and it has no fee. It will be used to arrange face-to-face meetings (as required by our agile approach) and to discuss possible reassignments and upcoming deadlines. Each member of the group will be able to see every message, preventing confusion on assignments and duplication of discussion. Discussion has been made about implementing audio communication tools such as Skype or Google hangouts as it has become clear that some members struggle with voicing their concerns over text based communication and members have found that giving direction over text-based communication simply creates more confusion. Therefore, during periods were team members must work remotely e.g. holiday breaks or visiting family, audio communication will take priority over text-based communication for discussing changes of roles. This coincides with the agile approach as we will reflect on how to be more effective and adjust communications tools accordingly. The group will also use email but this will only be used to arrange face-to-face meetings with customer and make small queries about ambiguous statements in the project brief. We decided to use email as it easily accessible and a universal mode of communication.

Due to the nature of the project, large amounts of code will need to be saved, edited and restored in numerous iterations. Therefore, we need to be able to store a backlog of code to ensure effective version control and to ensure all members of the team are collaborating on the most recent documentation. Version control is required for several reasons: to prevent concurrent work from conflicting; to enable the tracing of changes of code to find the root cause of bugs; to provide backup storage should code be lost or if new versions do not run. The tool we will use for this is GitHub. GitHub boasts other useful features, such as code review where members can request reviews of their code from peers and in return peers can comment in context within the code itself and propose changes [4]. The moderation tools, such as pull and conversation locking, will help the team to focus on coding [4]. A final useful feature the team will use is the project management feature. This tool allows the team to assign issue and pull requests to certain team members, making it clear who is doing what [4]. It will also allow the team to keep track of tasks by creating milestones which can be selectively ticked off to measure progress

A further Git tool will be used to help team members new to Git get to grasps with branches, commits and merges. The tool we will be using is GitKraken. Several team members found it difficult to use git commands within the command window and due to this the team sort an appropriate tool to help struggling team members. Git Kraken will help our team be more productive by enabling members to visualise and see the merging of branches within our git project and have a greater understanding of our version control system. This tool will help our team to work more collaboratively by ensuring less technically skilled members still have the opportunity to provide vital input on the implementation of our game.

Throughout the SEPR project, a key deliverable at the end of each phase is the presentation of the report. Due to this, the team needed a collaboration tool which can allow the sharing, downloading and editing of text documents in real time. This is especially vital as team members have different working schedules, and the Christmas break will mean the team cannot work in a face-to face working environment. Google Docs was selected as the appropriate tool and is justified as all members were familiar with its support of intuitive real-time editing of documents. Its integration into Google Drive also meant that document storage and document collaboration could take place within the same environment, minimising the time spent learning separate tool protocols. Google Docs also enables the final report to be exported as a PDF meaning that compatibility issues can be avoided when switching between document production tools to export files.

A key architectural component of our software project is UML diagrams. Therefore, we need a free tool which can provide intuitive and collaborative resources for diagramming and the ability to export these diagrams to other file

types [5]. For this we will use an online tool called Lucid Chart, as it provides all previous requirements, a robust backup system for our documents and an extensive shape and connector library [5].

Team Organisation

As mentioned previously, each member of the group is of a similar experience with software engineering projects, and are all seeking personal development. Subsequently, we decided that no one member of a group will be responsible for the whole of a particular task throughout the project, but each member will be given the opportunity to provide input on every deliverable. This should help to fuel collaboration and creativity as each member's skill set can be applied to each deliverable, producing a more well-rounded product. However, the team discussed that each work-package of a deliverable should have an 'expert' who makes executive decisions on their allocated work-package. This was done to ensure that each component fits the specification in the product brief, the demands of the customer, and upholds to the team's agreed quality standards.

In line with our agile method, our team is self-organising- which helps us to share the workload more evenly and make accommodations for members who may feel they are better suited to other tasks. Taking this approach means that the team can make adjustments to current organisation as we see fit to produce the best product. These adjustments to team organisation are discussed at our two weekly face-to-face meeting (in line with our agile method) where further discussion on tasks that have been completed, tasks to be completed, and potential risks that may interfere with progression take place. At the end of each meeting, a to-do list is updated and allocations of tasks for the next iteration are noted.

Role and Responsibilities

An important part of an efficient team is the allocation of roles and responsibilities. A role is a task that needs doing and without clearly identifying the responsibilities a team can become confused about what actions need to be taken. The following roles were allocated for Assessment 1:

Role	Responsibilities	Current Individual	Reasonings
Meeting Chair	Organising when and where meetings will take place and basic structure of discussion	Tom Burroughs	Previous experience with role with HACS module and strong communications skills.
Report Master	Overseeing of the production of the report, allocating members with sections of the report and focus on good English/structure.	Joe Frankish	Previous experience with report write-ups from HACS module and good literacy skills.
Software Manager	Responsible for executive decisions of architectural design and code implementation.	Huw Christianson	Most coding experience in the group and has worked with game development before.
Documentation Manager	Organising and holding of documentation for project.	Isaac Lowe	Previous experience with Google Drive/Git Hub and good organisation skills.
Risk Manager	Observing potential risks and ensuring risks are controlled and monitored.	Suleman Zaki /Beatrix Vincze	Both have observant natures and looking to improve on their analytical skills.

The roles stated above are more of an abstract rather than concrete nature, but roles with more concrete responsibilities- such as programmer and tester- will become more prevalent during the project's different phases. These roles and the previously mentioned roles were allocated after an open discussion between the team where members' skill-sets and experience were discussed, roles were then allocated based on these considerations. However, roles can be changed to allow for changing requirements, or the need for personal development (members wish to improve coding and leadership skills), or potential long-term absences.

The assigning of the same role to numerous team members provides unnecessary confusion and potential conflicts, mainly arising from team members working on the same piece of work but not collaboratively. As a result, roles such as programmer and tester need to be broken down further to avoid such miscommunications. For example, the tester role has been broken down into White box testers and black box testers who are responsible for the associative testing method. This break down was repeated with the programming role but with the obvious changes. This has also lead to the fulfilment of our need for personal development. For example, sub-roles within key roles have

a varying skill requirement allowing less skilled team members to gain experience in an unknown role. As an example, black box testing requires very little programming skill and no knowledge of the implementation architecture so less technically skilled members can still gain experience testing software.

As a team, during the storming process, we made no allocation for the team leader role, as we felt a natural leader will emerge. On discovering feedback from previous SEPR groups it became clear a leader was needed to mediate problems such as underperforming members. Due to this a leader was selected based on previous experiences leading groups through software projects and willingness to take on a more prominent role. A careful selection was made to select a member who had the confidence to hold members who are underperforming accountable and regulate workloads to ensure deadlines were met. Despite this, a clear distinction between the leader and 'experts' for each workload was maintained by privileging 'experts' with executive decisions.

Project Plan

Detailed Plan for Assessment 2

After the admission of the deliverable for Assessment 1, the group will meet twice between the 7th November and 10th November to discuss the work-packages and requirements for Assessment 2. This will be done to make sure all team members have a full understanding of the tasks, the timeframes for each task, and the changes to team organisation. This time will also be used to create suitable coding standards for the implementation process, to ensure all members with the same roles e.g. programmer abide by the strict standards and that coding appears consistent and readable.

Implementation will begin straight away (9th November) once the initial meetings have ensued and finish the week before the deadline (14th January). The design of game graphics, such as characters and map features will commence from the 9th November to the 10th December as this component is independent and not reliant on the other implementation components. We will use the paper prototypes, created by each team member in Assessment 1, as basic templates for the graphic designs of key features. Programming will commence the week after on the 12th November once a further meeting has taken place to make allocations for roles surrounding programming. Discussion on team members' programming skill-sets and plans for personal development will also take place at this meeting to select who will be most suitable for which programming tasks. The team has planned for all programming to be completed within seven weeks, finishing on the 24th December, so on returning from the Christmas holiday period, efforts can be focused on documentation and testing. Seven weeks have been allocated to programming as we feel that this is enough time based on the programming skills of the team members, but we will be flexible with this as some members may take on too much work. The documentation of code will run alongside programming, but one week has been allocated after programming has finished to make adjustments to the documented code. The later stages of the implementation work-package will only involve adjustments to documented code and the write-up of features not fully implemented. This will take place between the 31st December and 14th January.

Software testing cannot commence until coding has begun and has been allocated nine weeks in total, starting the 19th November and ending the week commencing the 14th January. The first software testing stage will be focused on planning and finalising our approach to testing. Once this stage has been completed, the last five weeks of the assessment will be allocated for the direct testing of the code and the documenting of test results. Software testing has been allocated nine weeks, due to the fact that our agile method measures working software as the primary measure of progress and therefore the team will focus on producing usable, tested software. As our Gantt chart shows (link at bottom of page) software testing and programming will run simultaneously throughout Assessment 2, although software testing will finish later. This is because of our agile approach. Using the scrum method we will make numerous iterations where code will be produced, documented and tested and then the next iteration will begin where amendments and changes to previous code will be made. Unit testing will commence simultaneously with the implementation process and will be undertaken by testers who have an insight into the architecture of our game code. This fits nicely with our agile methodology as after each scrum Units tests for particular classes or functions can be created and as more features are implemented Units tests can be built upon and sequenced. On the other hand, black-box testing will begin the week commencing the 14th January as this requires a more finalised implementation. This will be carried out by testers who were not responsible for implementation.

The architecture report is dependent on producing working, tested code and therefore work should not begin on this work-package until programming and testing are near completion. Due to this, work on the architecture report will begin on the 17th of December with the plan to have it finished the final week of the assessment.

Updates to the Assessment 1 deliverables and the website are of a medium and low priority, respectively. Due to this, they have been allocated the least amount of time. The website will be updated on the final week as all work-packages must be completed before updating can take place. Updates to the Assessment 1 document will mainly be made during the last two weeks of Assessment 2, with method and planning, requirements and risks planning and mitigation all being allocated two weeks. A purposeful effort was made to allocate the last two weeks to updating the deliverables for Assessment 1 to avoid unnecessary revisions, as changes to team organisation, planning and risks are likely to occur throughout Assessment 2.

Detailed Plan for Assessment 3

Before the implementation for Assessment 3 takes place, the group will attend the compulsory SEPR practical the week of the Assessment 2 hand in. Members of the team will make key notes on other SEPR group's presentations, with careful considerations being made on whether other group's deliverables match our expected quality and whether the amount of effort needed to fulfil the full specification is possible within the short time window before Assessment 4. Further details will be noted on how clear each group's requirements specification, architecture, testing and implementation report are. The last point will be of the most important as the more clarity we get with previous methods the more easily we feel we will be able to integrate and begin making changes/improvements.

On the week commencing the 21st Jan, after that week's SEPR practical (described above), the team will meet to discuss the potential group transition. Each member of the team will put forward which team they believe we should select. A discussion will then commence deciding which team to select. Should a unanimous decision not be made the group leader will make the final executive decision and will then email the lecturer with our choice. Another meeting will then be arranged for the week commencing the 28th Jan where team members will allocate potentially new roles and the responsibilities for these roles will be laid out. Plans will be made about the changes which need to be made from the previous group's method and planning section, for example, their Software Engineering methodology or implementation strategy. Within this meeting the group will also voice any concerns or issues which developed during the end of Assessment 2 period.

From the two meetings, the team will have a suitable list of implementation features which will be implemented in order to meet the full specification of the game as laid out by Assessment 3. This means that implementation will begin the week commencing the 28th Jan rather than the 21st Jan as previously planned. This is to allow time for more exact planning of the implementation or team organisation. Similarly to Assessment 2, testing will run simultaneously with implementation as discussed in our agile methodology. Although, black-box testing will be left until the later stages to allow for more efficient testing. Implementation will finish the week commencing the 4th but this will likely spill over due to changes in team members circumstance.

A major change to planning which will be employed for Assessment 3 is working on write-up deliverables concurrently with implementation and testing. At the end of Assessment 2, it has become clear that the writing up of all documentation at the end has lead to unnecessary stress and the rushing of these deliverables. In Assessment 3 write-up deliverables, such as the change report, will be worked on concurrently with implementation and testing with finalisation made at the end as planned. Work on write-ups will take place between the week starting the 28th Jan and 11th Feb.

A Gantt chart has been created to give a systematic plan for the rest of the SEPR module. The chart shows the amount of work that needs to be done in certain time periods as described previously for Assessment 2 and 3 as well as task dependencies (shown using black arrows). Using the chart, all team members will be able to easily identify how much time has been allocated for each work-package, hopefully, minimising the risk of delays. Despite this, the team will be flexible and will likely make changes to the plan. Potential causes of changes to the plan include a greater understanding of the tasks in Assessment 3, improved skill-set with development tools, reducing the timed needed for each iteration and unforeseen absences of team members. This became particularly clear towards the end of Assessment 2 as the team became much more familiar with our agile methodology and hence worked better as a team. This allowed sprints to be more efficient and more productive. We believe that this was due to our team being able to more frequently share concerns, minimising the time spent brooding over mistakes rather than finding a viable solution. A perfect example of this was the misallocation of roles. One team member felt that the Unit Tester role allocated to them was out of their technical depth and therefore, the team adjusted roles within the two weekly meetings. As a result, small changes to the plan for Assessment 2 were made to allow for such a transition. Implementation time was extended to allow some programmers to both program and Unit test their code rather than having completely separate testers. Updates to deliverables where also brought forward to allow members who had finished or changed section to provide further input. Despite this, an effort was always made to direct back towards the critical path.

URLS to Updated Gant Charts

Updated Gantt Chart: https://teamcraigzombie.github.io/assets/downloads/UpdatedGanttChart.pdf

Updated Gantt Chart with critical path: https://teamcraigzombie.github.io/assets/downloads/UpdatedGanttChart-CriticalPath.pdf

References

[1] Itinfo.am. *Software Development Methodologies.* 2018. Available at: http://www.itinfo.am/eng/software-development-methodologies/

[2] I. Sommerville. *Software Engineering*. Pearson, tenth ed., 2016, pp. 56-63

[3] M. Beedle et al. *Manifesto for Agile Software Development.* Feb 2001. Available at: http://agilemanifesto.org/

[4] GitHub. *The world's leading software development platform - GitHub* . Available at: https://github.com/

[5] Lucidchart. *Online Diagram Software & Vision | Lucidchart.* Available at: https://www.lucidchart.com/