1.4 | Software Engineering Models, Methods and Tools

1.4.0 | Introduction and Outline

Within an engineering task of any scale, it is wise to consider the processes and engineering methods through which the project will be completed. Using methods should combat the proclivity for complex team-based tasks to tend towards disorder in the absence of well-defined plans and aims. In response to the game-development brief, and with this mind, this white-paper will document the processes, methods and tools *Un-Two* intends to use to deliver on the client's brief.

Un-Two have chosen the Agile development paradigm to guide the principles of the project's software development. Project development methodology will be guided by the RAD (rapid-application development) approach. In brief, we believe the core tenants of agile – namely its flexible response to change and efficiency with small teams (both stemming from its non-deterministic and non-linear nature[1]) – are most appropriate for a small-scale videogame design project: *The Mysterious University of York*.

Game development shall take place in *Unity* and will make heavy use of the engine's integration with *Visual Studio* for *C#* scripting. *Visual Studio* provides native support for *Visual Studio Team Services* (VSTS), Microsoft's version control system. It is through this that the team will code collaboratively. As such, the code repository will be hosted on a *visualstudio.com*.

Aside from group meetings, discussion pertaining to both project and group-meeting planning will be conducted over Facebook Messenger. Documentation is worked collaboratively via Google Docs, where files are collated within a single virtual drive, promoting both orderliness and easy-of-access.

1.4.1 | Software Engineering Method Justification

The relatively short-time frame of the project coupled with the size and inexperience of our team quickly ruled-out many methods that large, established industry firms may consider. Furthermore, our project is a greenfield project, meaning that we cannot rely upon existing methodologies and must forge our own.

Examples of methods include the widely-used *Scrum* method. *Scrum* emphasises bursts of rapid development within a longer timeframe, which is something that does not excel in our circumstances. Given the exceedingly short time-frame by software engineering standards, we doubt that using the *Scrum* method will encourage enough development to deliver a completed product.

The *waterfall* engineering model was also considered. This model breaks down the development timeline into discrete phases which are linearly adhered to. As such, a *waterfall* method does not allow for testing to be integrated into the development phase. Rather, it is done all at once towards the end of the project, which in our timeframe, would not provide enough time to correct all of the issues.

Resultantly, the RAD method has been selected because it enables us to work most efficiently, given the client's loosely defined requirements and time-frame. RAD also excels where frequent client meetings and review of product are possible, and this something available to us during the development process. In fact, regular client meetings ensure that the end product fully agrees with the client's requirements and being able to test continually leads to quick bug identification and streamlining of development.

Furthermore, RAD puts less emphasis on planning and more emphasis on process. Such a focus is particularly relevant given the lack of established practices and common engineering methods amongst the team. The majority of work done in this process will be completed collaboratively in different locations. In this case, RAD's rapid construction technique is extremely useful by letting multiple people update and produce code in quick succession to each other.

The one possible downside of RAD is that it requires a higher level of competency (there is little pre-defined structure and methodology) from each team member than other methodologies[2]. In our team we have three

members with a higher level of ability in game development. To compensate for those without, our original planning session will be split the tasks in our driven plan into phases and assign specific tasks to each member based on ability, while still maintaining a fair split in the work. This plan should successfully harmonise RAD, and encourage fast code delivery with a discrepancy in team competence.

1.4.2 Engineering Tools

Game Engine: Unity

Unity has been chosen for the following reasons: Unity's scripting language is C#. Of the six group members, three have existing competency in C#, and all have experience with Java (a decidedly similar programming language). With existing experience, the team will be able to devote more time to producing client deliverables, rather than learning to use a new technology. Next, Unity provides pre-existing tools and features (such as animation support, graphics rendering, object transformations) that will allow the team to focus upon implementing gameplay features rather than technical dependencies.

Version Control: Visual Studio Team Services

VSTS provides all features that one has come to expect in a version control system but its main strength - as previously discussed - is its integration with Visual Studio; the application where coding for the game project will take place. A noteworthy feature of VSTS (beyond other VC services) is its stakeholder support which provides the client with instant access to project status, and an ability to provide direction, feedback and approve releases [3], should they wish to.

IDE: Visual Studio

Visual studio is a development environment used for programming in languages such as VB and C# making it ideal for our use as we are going to be programming in C#. This is the best IDE for C# as it has a lot of inbuilt functionality such as intellisense and other useful organisational features for the code. In Visual Studio there are a lot of shortcuts and templates available meaning that it will be easier than having to type all generic syntax such as the structure of a class declaration or inherently necessary library imports. Visual studio is a much better choice than other development environments such as notepad++ because it is created with developers in mind and has lots of useful functionality which makes development faster and more efficient.

UML Tool: StarUML

UML has been used to create the soft architecture diagrams and we will continue to use the technology for future UML diagrams throughout the project. Our choice of UML software is StarUML. It is free, has a simple and easy-to-use interface that members of the team can quickly learn and supports multiple file types (png and pdf) for export so that diagrams can be quickly and easily uploaded to Google Drive (our chosen collaboration platform) and viewed there natively, without the need to download and open them in StarUML.

Communication: Facebook Messenger

Given Facebook Messenger's integration with team mobile phones, it offers the fastest notification between any possible communication services. Dedicated communication software such as *Discord* and *Slack* were considered, but multi-chat room functionality is not needed given the small team size. When new messages are received, team devices are pinged, ensuring that team members can review messages as soon as possible, facilitating a fast alleviation of issues (this is congruent to the RAD methodology and the given timescale).

Documentation: Google Docs

Google Docs acts a private virtual drive, accessible by all team members. It is hoped that its ease of use and effortless collaborative nature will minimise any potential impact on the software engineering phase. Indeed, Google Docs is universal and has little-to-no software or OS dependencies. It provides a non-destructive commenting system for rapid review of documents and a version history acting as an automatic backup.

1.4.3 | Team Organisation

Un-Two has adopted a flat management structure for the project, where individual members will be be responsible for the delivery of certain tasks. This works well with RAD's method of independent work or work within small groups.

We believe a non-hierarchical management approach will work with our team because conflicts and inquiries can be resolved very quickly. In short, we believe any form of middle management is redundant given the team size, the scope of the project, and team personalities. Where conflicts in decision may arise, a democratic process will be undertaken to resolve such conflicts.

As stated, individual members are assigned responsibility for some critical tasks. Other members can contribute towards documents using *Google Docs* or code through *Visual Studio* but the assigned person remains liable for its delivery. It is hoped that such an approach will maximise collective input into the work from varying members of the team meanwhile maintaining an incentive for it to be completed and delivered. This way, the project is delivered in full despite any managerial authority. We believe defining strict roles for team members would only diminish team flexibility

Regarding the deliverables within the software engineering task (assessment 2), certain members will be given priority (not to be confused with responsibility) input into the task according to enthusiasm and ability, although it is too early to provide details about such priority. Such decisions will be made at a time when they are required, in accordance with RAD methodology. Our gannt chart presents some initial plans for the first section of work. Here we can see the intention to have no more than three tasks going on at any one time, which allows us to work in small groups of two or three and mitigate risk to dependency-heavy code sections.

With no central person acting as a go-between amongst team members, group meetings and communication become doubly important as each team member needs to relay their progress back to the team as a whole. This has prompted the need for regular meetings that satisfy both the absence of leadership and the need for adaptability within the *Agile* paradigm. Throughout development there will be ongoing peer review, looking at each finalised piece of code. Review meetings will be set-up at each progress step and for each finished piece of code. At these meetings, members who were not involved in writing the code, will be selected to review, test and relay back to the original author changes to the code.

1.4.4 | Plan for the rest of SEPR

Our plan for Assessment two is detailed in a Gannt chart (available on our website), with higher priority tasks starting first, to allow for delays in these tasks to happen and not affect the final deadlines. This plan is detailed below:

Task	Start Date	Deadline
Code	22/01/2018	29/01/2018
Documentation	29/01/2018	05/02/2018
Justification of Architecture	29/01/2018	05/02/2018
GUI Report	05/02/2018	11/02/2018
Testing Report	05/02/2018	11/02/2018
Methods and Plans	05/02/2018	11/02/2018
Risk Assessment	05/02/2018	11/02/2018
User Manual 3	05/02/2018	11/02/2018
Write up	11/02/2018	18/02/2018
Code	19/02/2018	12/03/2018
Documentation	12/03/2018	19/03/2018
Testing	19/03/2018	26/03/2018
Traceability report	02/04/2018	09/04/2018
Requirements report	09/04/2018	14/04/2018
Catch up	14/04/2018	23/05/018
Presentation	23/05/2018	01/05/2018
Write up	23/05/2018	01/05/2018

We plan to update this table with more detailed plans once we have further details of Assessment 3 and 4 requirements and deliverables.

Our Critical path and task dependencies are shown in our Gannt chart, with more critical elements being started sooner than non-critical elements so that the project's foundations are created first. Code is divided into 4 areas which can be developed independent of each other due to our use of object oriented methods. Tasks have been split such that efforts can be focused on a small amount of tasks at any one time in order to minimise delays and prevent a myriad of half-completed tasks, leading to project failure.

Reference

1. Mitchell, Ian (2016). Agile Development in Practice. Tamare House. p. 11.
2. https://airbrake.io/blog/sdlc/rapid-application-development
3. https://docs.microsoft.com/en-us/vsts/security/get-started-stakeholder