DEPARTMENT OF COMPUTER SCIENCE

THE UNIVERSITY OF YORK

SEPR - Assessment 3

# Updates on Method Documentation

Evil Geese (Un-Two Takeover)

## Contents

# 1. Agile Incremental Development

An agile, incremental development process is the most appropriate general process model for this task [1]. This is because our requirements are likely to change during our project due to changes in the customer's needs and revisions in our design solutions. Consequently, it is important that we are flexible in order to prioritise our requirement implementation whilst still completing our project within the limited time frame. We must therefore interleaf our requirements engineering with our design development and allow both to evolve simultaneously as specified by an agile development process [1].

This differs from the other agile methodologies, whereby requirements are explicitly stated at the beginning of the process and if changed, require the entire system to be redesigned. Our engineering method will also contrast the plan-driven waterfall model as it is easy for us to meet with the customer during our developmental cycles, thus allowing the customer to have more opportunity to evaluate our product in real life: Hypothetically, this agile method should prevent the common misunderstandings caused by the miscommunication of the system to the customer through documentation alone in the waterfall model [1][2].

Being flexible and embracing change in this agile way allows the customer to have more input into the design development, since modifications can be made at every increment. If the customer is dissatisfied with our present system or if changes need to be made, only the current increment has to be changed - this is a huge benefit to this incremental agile method, as it will reduce the amount of wasted work, which is crucial given our short schedule [1][2]. The rapid pace of iterations of this method will work well for our project as our group is small; this means we will be able to communicate frequently and informally, which will allow for frequent, collaborative work on iterations of code and accompanying documentation.

# 2. Development Tools (Updated)

In order to maintain an open workspace and collective ownership of the product, it is important that all of our development, collaboration and communication tools equally are available to all members of our team (including our customer) and make it easy for our group to pass our project on to a different team at a later stage. This means the tools have to be free for an unlimited time, and must be easy to learn how to use.
We have set out our development tools in Table 1 with the following categories:
Tool Demand: Features a tool is required to support.
Team Needs: How we require the tool to operate specifically for our group.
Tested Tools: List of tools that have been trialled by the group for this demand.
Justification for Tool: Reasons for using the tools.
Associated Risks: Potential problems with tools and associated risks in our risk register.
Current Tool: The tool currently being used by our group for this demand.

Table 1: Development Tool Table

| Tool Demand | Team Needs | Tested Tools | Justification for Tools | Associated Risks | Current Tool |
|---|---|---|---|---|---|
| Communication | A tool to enable us to communicate when we are not together. Allow rapid responses between group members as well as allowing group members to see all interaction between all other members. | 1. Facebook Messenger | 1. Free, available to all group members using University accounts. Easy to use. Reduces duplication of discussion and therefore helps to alleviate miscommunication. Same information is accessible to everyone at the same time. | 1. Risk that members do not use or regularly check this tool and miss out on vital communication. | Facebook Messenger |
| Storage | Communally accessible place to store group document such as notes, drafts and deliverables. Instantly updated so that everyone can access the most recent version. Available to use anywhere. Store a variety of file types. | 1. Google Docs | 1. Free, available to all group members using University accounts. Easy to use. Allows simultaneous editing and useful commenting tool as additional communication about specific parts of documents. Everyone has access to everything, therefore minimising duplication and miscommunication. | 1. TP1 - catastrophic if storage is lost and work cannot be retrieved. | Google Team Drive |
| Version Control | Allows all members equal access. Allow previous versions of our code to be retrieved. | 1. Github | 1. Free, available to all. Minimise duplication of information so that no manual updating of code is required and to reduce errors be ensuring that everyone is always working on the most updated version of documentation. If work is lost or if new versions do not work, previous versions can be restored, also useful for future groups to pick up our project. | 1. Risk that work is lost if new edits aren't committed. | Github |
| Documentation | Real-time editable text editor, accessible by all. Documents can | 1. Overleaf<br>2. Google | 1. Free, available to all. Enforces uniform style. Built in-version control allows | 1. Steep initial learning curve as must learn to use LaTex. | Google Docs |

| | | | | | |
|---|---|---|---|---|---|
| | be exported as PDFs. | Docs | previous versions to be restored if necessary. Easy to create bibliographies to reference materials.<br>2. Free, available to all. Easy for group members to use, easily integrates documents into Team Drive storage. Easy for new groups to pick up and use as it is a familiar system. | Risk that other groups may be put-off from our project if unfamiliar with LaTex.<br>2. No enforced uniform style - deliverables may appear less professional or be confusing. No simple inbuilt referencing system. | |
| Activity/Class/UML Diagrams | Cloud based, free. Ability to save and store diagrams to reduce duplicity. | 1. Lucid Chart | 1. Contains full set of UML shapes, allows multiple charts to be created. | No risks noted | Lucid Chart |
| Planning (Gantt Charts) | A free tool to easily create, edit and export Gantt Charts. It must show work packages and tasks. Allows shared access. | 1. Team Gantt | 1. Free, allows some degree of collaboration. Editable charts allow tasks to be changed and time frames to be edited as assessment progresses. | 1. Not all members can access this site for free and so only 3 members are able to edit the chart - this can be mitigated by having regular meetings to assess this chart and edit the chart as a group. | Team Gantt |
| Development Environment | Free, well-documented, portable. Able to export to different devices. Compatible with common platforms. | 1. Visual Studio<br>2. MonoDevelop | 1. Allows frequent testing of program.<br>2. Built-in to Unity therefore convenient to use. | No risks noted for either | Personal Preference |
| Game Engine | Free, easy to use, accessible to all members of the group. Works with version control tool. Supports 2D. | 1. Unity | 1. Will ease transition for other teams if they are also using unity. Existing team member experience. | 1. Other groups might not know | Unity |
| Art Design Tool | Tool to use to create scenes, sprites and buttons for our game. Simple, not time intensive. Should create images with sufficient detail. | 1. GIMP<br>2. Photoshop<br>3. Tiled<br>4. Aseprite | 1. Free, easily available to all, very basic tool.<br>2.<br>3. Allows tiles to be created for ease of further scene creation with Photoshop. Helps to keep consistent art style and will ease further scenes made by future groups. | 1. No familiarity with this tool<br>2. Restricted access may limit speed of work. Possible learning curve risk for unfamiliar users.<br>3. Might not be appropriate for all sizes of tile - some assets may need to be larger than the | 1. Photoshop<br>2. Tiled<br>3. Aseprite |

| | | | | | |
|---|---|---|---|---|---|
| | | | 4. Used to create character sprites at different angles. | given tile size.<br>4. Not free, risk that next group may struggle if they want to introduce new characters therefore this is a possible deterrent. All 6 (requirement F1) of our characters are already implemented. | |

# 2. Team Organisation

## 2.1 Justification for Organisation (Updated)

We will not be using a hierarchical structure since every member of this small group has similar relevant experience and there is a lot of work to complete, each member will need to be equally involved in each iteration of development across all roles. This approach also extends from the principles of the agile method that we are adopting by using the collective ownership strategy so that every member will be responsible for at least an element of every deliverable in the project [1]. This avoids allocating entire individual tasks to a single member of the group which should help to prevent the group from becoming entirely dependent on one member all of the knowledge on a specific area of the product. If not avoided, this could lead to problems with communicating the product to the customer; it may put a task at risk if the sole contributor becomes unavailable; and may waste large amounts of time if that individual needed to explain their work to the rest of the group. This would be particularly problematic given our short time frame for the project. However, if everyone is collaborating on the same task, it may be difficult to decide which ideas are implemented. Therefore, it is best if every main deliverable has a unique leader to make the final decision about that task and ensure the quality of the task they are responsible for - making sure that it fits the specified requirement from assessment documentation and stakeholder demands.

We have found that being a self-organising team has been very successful as this means that we each manage our own workload and share work amongst ourselves if we find that someone else may be better suited for a task, or if we have allocated too much to ourselves. Over the winter break this was particularly useful as we all had to balance this project with revision, so long as we stuck to the approximate timings of the Gantt chart, we were all able to produce work and discuss any issues over Google Hangouts.After meeting the group, at the start of assessment 2, we understood more clearly what was required to build the game that we envisioned. From this, we were able to expand on the vague tasks, which would need to be carried out. .Due to the change in nature of our project there would be some new roles and responsibilities to be assumed, in terms of allocation, having a meeting where we as a team can discuss which task would be suited be suited for an individual also identifying overlaps in skills would mean generally, a task wouldn't be done by one person . But as we do progress throughout the project new tasks do occur so ensure we continue to be organised we have the group manager assess the task and determine the correct member to carry it out.

## 2.2 Role Allocation

We decided that each main component of the game should have an owner to manage associated tasks so that everything is owned by somebody so no tasks or components are neglected. As we are using an agile methodology, we are following a self-organising structure that allows team members to find a contributing role that fits their prior knowledge, skills or desire to improve a certain ability. These reasons are given in the team role table (Table 2) under 'Reason for Allocation'.

Table 2: Team Roles

| Role | Responsibilities | Individual | Reason for Allocation |
|---|---|---|---|
| Website Manager | Making sure the website is readable, usable and updated. | Owen Pantry | Interested in web design, desire to use and improve website design skills. |
| Group Manager | Arranging regular meetings, assigning sub-roles to members for work-packages as they arise, in charge of project plan (Gantt chart). | Callum Deery | Good organisational skills and takes control when the need arises. |
| Documentation Manager | Ensure formal documentation fits the specification of the assessment. | Michael Carleton | Good at checking grammar. |
| Requirements Manager | Ensures that any changes in requirements are implemented in the project and responsible for checking that the project to meet minimum requirements. | Luke Needham | Read the applicable literature and has good social skills and a desire to liaise with the customer. |
| Risk Manager | Ensures that risks are monitored, maintained and controlled throughout project. | Kieran O'reilly | Observant character with good analytical skills. |
| Software Manager | Has final decisions in code creation and architectural design. | Alasdair Kirby | Best previous experience with programming, including experience with potential middleware, Unity. |

# 3. Project Plan for Assessment 4 (New)

The timeframe for assessment 4 encompasses the Easter holiday in addition to 6 weeks of term time, which provides the team with more time to implement the necessary deliverables than assessment 3.

On inspection, many of the deliverables of assessment 4 seem similar to those of assessment 3. In this case, we can use a similar timeline as we did in assessment 3 for deliverables 2 & 3 of assessment 4.

We assume that most, if not all, game sprites will be present in the game, which removes a need to spend time creating new resources. Further, the code has been extended by a team before us and therefore we also assume that finalising the game should be more straightforward than if the game was developed by a single team, without regard to extensibility. The storyline should be established and likely finished by the previous teams; again, reducing potential workload.

We hope to finish coding by the middle of the Easter break, as this is a prerequisite for all other assessment 4 deliverables. Doing so provides 5/6 weeks to create the architecture report, followed by the testing report and final the project review report. This order of documentation has been chosen given the seemingly natural state of document dependency. We believe one week for each of the documentation deliverables is ample, and will provide a two week period to amend any of the documents (perhaps for greater cohesion) and work on the client presentation.

# References

[1] I. Sommerville, Software Engineering. Pearson, tenth ed., 2016.

[2] M. Sami, “Choosing the right software development life cycle model,” March 2012. [Online]. Available: https://melsatar.blog/2012/03/21/choosing-the-right-software-development-life-cycle-model/. [Accessed: 3/12/17]