Project plan + study diary

Jungle Hunt: Revived

version 2.0

TUT	Pervasive Computing	TIE-21106 Software Engineering Methodology
Author: Markus Ylisiurunen		Printed: 29.04.2018 20:02
Distribution: Markus Ylisiurunen Tuomas Pekkanen Jere Metsäranta Pedram Ghazi		
Document status: Done		Modified: 29.04.2018 20:02

VERSION HISTORY

Version	Date	Authors	Explanation (modifications)
1.0	28.01.2014	Marko L.	Initial version
1.1	11.02.2014	Marko L.	Deleted finnish text
1.2	18.01.2015	Tensu	Sections 1.4.x, cosmetic tuning
1.3	26.1.2015	Marko L.	Final toucher
1.4	16.01.2017	Kari S.	Adaptation for 2017 needs
1.5	08.01.2018	Farshad A.	Adaptation for 2018 needs
1.6	08.04.2018	Markus Y.	Add 3rd sprint's material
2.0	29.04.2018	Markus Y.	Add 4th sprint's material

TABLE OF CONTENTS

1. PROJECT RESOURCES

This chapter holds the project resources.

1.1 Personnel

Tuomas Pekkanen, Product Owner
Has prior experience in Unity, C# and C++ in small scale projects and from a summer job. Interested in developing games, optimizing code and technology in general.

Estimated contribution: 35 h
Absences: None

Contact information:
Email: tuomas.pekkanen@student.tut.fi

Markus Ylisiurunen, Team Member
Is familiar with C++, Python, JavaScript, PHP, Ruby. Has made a few side projects on his free time. Works as a software developer. Interested in the web as a whole.

Estimated contribution: 35 h
Absences: None

Contact information:
Email: markus.ylisiurunen@student.tut.fi

Jere Metsäranta, Scrum Master
Prior experience limited to C++ and Python. Interested in learning new stuff, also majors in programming. Has one website project going on free time.

Estimated contribution: 35 h
Absences: None

Contact information:
Email: jere.metsaranta@student.tut.fi

Pedram Ghazi, Team Member
Preferred language for coding is Python but also knows C++, Matlab and PHP. Is familiar with JavaScript. Right now, is working part-time as an RA and field of his work is machine learning. Unfortunately, he does not have prior experience in C# or Unity. He is also interested in AI and web development.

Estimated contribution: 35 h
Absences: None

Contact information:
Email: pedram.ghazi@student.tut.fi

1.2 Process description

We had our first meeting at 16.1.2018 and we decided to create a Slack team where our group can communicate, ask questions and manage the project. We plan on relying on online communication for the majority of our communication needs. Whenever the need arises, we can meet face to face and discuss the current issues as a group. We have agreed to have a weekly online meeting at 18:00 every Tuesday.

At the very beginning of each sprint we will decide new roles for every person and which features belong to the next sprint as a group. By rotating the roles, we hope that everyone will get the most out of this project. After that we'll assign those tasks equally to each member and everyone can work on their own time. This hopefully ensures that everyone will find the best time to work and it won't have too big of an impact on other things. Since we are using Slack as our communication method, everyone can ask questions at any given time.

We have defined “ready” to mean when a new merge request is submitted without WIP status. Once the merge request is reviewed and possibly fixed and everyone is happy with it, it gets merged to master. This is the point when a task is “done”.

We have decided to use the following branching model. For new features (customer requirements or tasks) we create a branch from master with the following naming convention: feature/<first name>/<feature name>. For fixes we create a new branch from master with the following naming convention: fix/<first name>/<fix name>.

Commit messages should always have a corresponding customer requirement or individual task. For customer requirements the commit message should follow the following convention: [CR<id>] Commit message goes here. For individual tasks the commit message should follow the following convention: [T<id>] Commit message goes here.

We build the game weekly for Windows and macOS. This is done manually because Gitlab’s CI/CD features are not enabled. If they were, we could build the game automatically on every update on master. Our methodology is that master is always working and it is always buildable. When one of our group member builds the weekly build, they just pull master to their computer and build the game to an executable from within Unity. Right now, the built executables are not stored anywhere online since the game is not distributed yet.

Our goal is to split the tasks and work hours equally on each sprint and finishing sprints on the anticipated time. We want to maintain high quality code which has went through a code review. We also want to have a high visibility of the changes so that every group member has a good idea of what is currently being implemented and what is yet to be done. We also want to adapt ourselves to the found issues and make changes to our workflow as the project advances.

Our project goal is finishing on time and getting a high grade which also means that we are aiming for a good workflow, communication and high-quality code. We will meet this goal by planning in advance, having good communication and clear roles in the project.

1.3 Tools and technologies

We’ve decided to build the game in Unity. We are going to use the latest version (2017.3.1) and stay with that to prevent any issues from using different versions. Tuomas Pekkanen has previous experience in Unity and he’ll be the person other group members can ask Unity related questions from.

We are using a repository hosted in GitLab to store our project. Each group member has access and can push new code. We'll utilize GitLab Merge Requests for code review. Each new feature/bug fix will go through a code review where at least one other group member will review the changes before merging to master. This way everyone can keep up with the changes and we hope to achieve better code quality.

Here are the relevant resources related to our project.

Repository: https://course-gitlab.tut.fi/sweng_2018/g06---ylo
AgileFant: https://app.agilefant.com/TTY-TIE/product/376284/tree

Table 1.1: Tools used in the project.

Purpose	Tool	Contact person	Version
Communication	Slack https://slack.com/	J. M.	-
Version management	Gitlab https://gitlab.com/	M. Y.	-
Development	Unity https://unity3d.com/	T. P.	2017.3.1
Project management	Agilefant https://www.agilefant.com/	P. G.	

1.4 Code refactoring

Code refactoring was done in merge request 34 (https://course-gitlab.tut.fi/sweng_2018/g06---ylo/merge_requests/34). The idea was to go through all of our scripts and fix issues related to comments, formatting, naming and the actual logic. All of these were done in the mentioned merge request.

This improved our code quality, readability and maintainability. We also were able to remove some unused variables and reduce the technical debt. We were also able to spot common issues (for example using a unnecessary function in Update()) and these could be avoided in the future.

It is very important to do code inspection and code refactoring in a larger project from time to time. Usually if there are some not optimal code, new features are going to be built on top of the existing code and it might (will) slow down the project and the code just keeps getting worse.

However, in our project, the lines of code are pretty low, and we don't have much technical debt. Even now, there were quite a lot of changes made to make the code better.

1.5 Code inspection

We did code inspection as well. It is documented in another file which can be found in the root of the repository.

Code inspection file: G06-code-inspection.pdf

1.6 Unit and integration tests

Integration tests and Unit tests are important for maintaining the quality of project. We decided to use Unity's integrated Test Runner, which handles running integration and unit tests automatically whenever a new build is made or when requested by the Test Runner window.

The tests include testing critical modules' compatibility (integration) and testing specific important functions in the most commonly used classes. For example, the story teller's functionality is unit tested and Game Manager is tested with other classes to make sure they work together.

1.7 Maintenance guide

Our maintenance guide is in an external file in the root of this repository.

Maintenance guide file: G06-maintenance-guide.pdf

2. STUDY DIARY

In this chapter, we review each sprint and write about our learnings during the sprint.

2.1 Sprint 1

2.1.1 What went well

We started to communicate early on and we got everyone involved and set up without major issues. We also discussed and tried to decide how we want to work and we decided the roles. Overall, we think the planning process was good.

2.1.2 What difficulties you had

One of the biggest resistance in getting the project started was the fact that we were using a completely new toolset. Only one member of our group had used Unity or C# before which made it quite hard to start to contribute to the project.

We didn't meet too often in sprint 1 since we thought that it'd be more useful for everyone to dig into the new tools by themselves and ask questions in Slack. That worked but it might have slowed down our progress a little bit.

2.1.3 What were the main learnings

We noticed that we should set more specific deadlines for ourselves so that we keep working during the whole sprint and not leave everything to the last week. We also noticed that keeping Agilefant up to date would help us during the whole sprint.

2.1.4 What did you decide to change for the next sprint

We decided that we want to meet face-to-face more often for short periods of time to discuss project related topics. This includes defining tasks, planning the workload and talking about possible issues we are facing.

2.2 Sprint 2

2.2.1 What went well

We met more often and communicated more in Slack. We also used Agilefant more and everyone got started with Unity and C#. We discussed problems and tried to help each other.

Our planning was also better than in sprint 1. This time we defined the tasks better in Agilefant and members started contributing to those.

2.2.2 What difficulties you had

We still started a little too late. We also had quite a lot of issues with Unity and Git which slowed us down. The issues were related to how Unity generates meta files.

We also had issues in general related to Unity and the development of the game, but we think that this will get better once we get better with Unity.

2.2.3 What were the main learnings

We learned to use Git branches better and the workflow got a lot better than in sprint 1. We also noticed how short meetings can help us to push the project forward and clear issues efficiently.

2.2.4 What did you decide to change for the next sprint

We decided that the most important thing to do better is to start early and get all the sprint related things planned right at the beginning. This way everyone has the time to do their part and we'll get more done.

2.3 Sprint 3

2.3.1 What went well

Everyone is starting to get used to working with Unity and we get more done in a shorter amount of time. We also did a cleaning sweep of our code so that it should be easier to continue from there.

We communicated well and created tasks at the beginning of the sprint.

2.3.2 What difficulties you had

We should try to schedule our time so that everyone is working at the same time on the game even if we are communicating through Slack. It's a bit hard to work since everyone was working at different (random) times.

2.3.3 What were the main learnings

Everyone got better at Unity and we also realized the importance of clean code since other people will also be reading it. From now on we try to maintain code higher code quality to save time in the future.

We should also consider having set times when all of us work on the project at the same time. Maybe a couple of hours per week.

2.3.4 What did you decide to change for the next sprint

We'll have a set time every week when all of us work on the project a couple of hours. This way we hope to spread the workload more evenly throughout the entire sprint. We'll also monitor code quality more closely.

2.4 Sprint 4

2.4.1 What went well

We had more meetings (retrospective on 11.04.2018, code inspection and one live coding session). We also were better at estimating the amount of work required to finish the remaining tasks and we read the necessary assignment information for this sprint earlier. Overall, we were better at managing the sprint and project.

2.4.2 What difficulties you had

Since this is the last period and we all had many other courses at the same time, it was difficult to find time to work on this and it also made it harder to work on the project at the same time. However, we were able to squeeze in a couple of sessions where everyone worked on the project at the same time.

Other difficulties were related to the actual coding but that's always the case with something new. However, on a positive note, this sprint was the easiest with Unity since we already had learned the basics.

2.4.3 What were the main learnings

It always seems like it's easy to manage a project but when you really try to do that it turns out being way harder than you would've thought. All of us got to be the product owner and scrum master once and that was really helpful for all of us. Now everyone has a bit of personal experience with every role.

During our live coding session, we tried pair coding. We wanted to see how it affects the productivity, learning and quality of code. Even though it might seem at first that it would be waste of time since two are working on the same thing, we noticed that it actually helps solve problems and the results are better. It might be something we'll try to do in the future as well.

Documentation is hard. It seems like there is always something left that you could write down and you have to find the balance of how much time you spend on documentation and the actual work. After we had written the necessary documentation, we had a discussion about it and the outcome was that it would have been easier to approach with a bit of a different angle. We should have planned the documentation beforehand and write down the details of each section at the same time they were actually done. This way the documentation would have been more mature and detailed. It would have also had better quality since not everything would have been done in one burst.

2.4.4 What did you decide to change for the next sprint

This was the last sprint but if there would be a next sprint, we would try to assign specific dates and times for our working sessions when everyone is online and works on the project. This was a little hard since everyone was very busy with other courses as well but in an ideal situation we would want to try to estimate the effort as precisely as possible and then based on those numbers we would schedule, let's say 120 % of the estimated effort, dates and times for the working sessions.

3. RISK MANAGEMENT PLAN

In this section, we go through the most probable risks that might affect us during the development of this project. Each risk will be rated based on probability and severity.

We will also analyze each risk and try to find a way how we can prepare for it. This way we hope that realized risks won't affect us much and we can keep working and making progress even if something comes up.

The following table collects all risks we have identified into one place. They will be analyzed separately in the following sections.

Table 3.1: Identified project risks.

ID	Description	Probability	Impact
PE1	Getting sick	2	2
PE2	Busy with other work	3	3
CU1	Project requirements change	2	1
TE1	New technologies take time to learn	3	3
TE2	New technologies don't work as expected	3	3
PR1	Scheduling issues	1	2
PR2	Not starting early enough	2	2

3.1 Personnel risks

This section goes through the risks related to individual group members.

3.1.1 PE1: Getting sick

It is quite likely that at least one group member will get sick during the project. This will decrease the time that the person can allocate to the project work.

Since we are working with software, it is not required to be present at some place at a specific time. Getting sick will most likely mean fever which will last only a few days. After the few days of sickness, the person can easily communicate via Slack and maybe even do some project related work.

We will try to avoid this from spreading to other group members by not having meetings where the sick person is present. We can inform that person via Slack and keep him up to date.

Root cause: A person gets sick.
Importance: 4
Avoidance: We will not be having meetings where the sick person is present.
Prevention: Depends on how serious the sickness is. We can discuss whether we need to redistribute the work assigned to the sick person if that's needed.
Recovery: Keep an eye on the incomplete tasks and other group members will take over if needed.

3.1.2 PE1: Busy with other work

We are all doing other course at the same time as this one. We might have other deadlines from other courses during our sprints and those might take a lot of members' time.

Three out of four members in our group are doing almost the same course with the same assignments which means that when there is going to be a deadline, it's going to affect the majority of our group.

We will try to communicate these deadlines early on so that we can prepare and do the work when we have the time. This will hopefully help us plan our workload better.

Root cause: Group member(s) have other assignments and have to spend time on finishing those.
Importance: 9

Avoidance: We can't avoid this completely, but we will try to plan our sprints with this in mind.
Prevention: Planning carefully and communicating the deadlines in a clear way to every member of the group.
Recovery: If needed, we will redistribute work or work longer days. With careful planning, we shouldn't be in this situation.

3.2 Customer risks

This section goes through the risks related to customers.

3.2.1 CU1: Project requirements change

The project is developed based on the requirements the customer provided in the very beginning. These might change during the project and it might render some already finished changes obsolete.

We are trying to prevent this by having a review meeting after each sprint where we go through the changes and the customer can tell us about possible changes they've decided to do. This way every sprint is locked, and the requirements can't change during the sprint.

This isn't too severe for us since the customer decides what they want, and we just deliver based on those requirements. This will postpone the delivery date but it's on the customer.

Root cause: Customer decides to change the requirements of the project.
Importance: 2
Avoidance: We have review meetings quite often so that the changes will be noticed early.
Prevention: This can't really be prevented.
Recovery: We will notify the customer about the possible impact of the change and update our tasks based on the changes.

3.3 Technology risks

This section goes through the technological risks.

3.3.1 TE1: New technologies take time to learn

This is probably the biggest risk we will be facing during the project. Only one member of our group has previous experience with Unity or C#. The fact that everyone else has to start from the very beginning is going to slow our progress significantly.

Our group members (excluding one) hasn't been doing much game development previously and this entire area of software is very new for most of us. This means that most of our time will be spent on researching and learning rather than actually contributing to the game itself.

There is no magical trick that would make the learning progress quicker and everyone just has to keep asking questions and spending time on learning. Tuomas will also help with Unity and answer questions.

Root cause: Group members don't have experience with the technologies used in this project.
Importance: 9
Avoidance: Cannot be avoided.
Prevention: Cannot be prevented.
Recovery: We have a group member who has previous experience and can help other members. Other than that, we just have to keep learning and spending more time experimenting.

3.3.2 TE1: New technologies don't work as expected

There is always the possibility that new technologies nobody has previous experience with will not work as they expected. This is inevitable and near impossible to find out before the actual work happens.

There are is no way to prevent against this because the technologies have been chosen for us and we can't use a technology someone has a lot of experience with. The only thing that could be done is extensive research before choosing to use a specific technology.

Root cause: Unexpected issues with new technologies
Importance: 8
Avoidance: By doing extensive research before starting to use it.
Prevention: By choosing a technology that someone has personal experience with.
Recovery: If the technology is popular, it is very likely that there are a lot of resources out there to help with the issues. It is also very unlikely that someone else hasn't had the issue already at some point and asked about it online.

3.4 Project management risks

This section goes through the risks related to project management.

3.4.1 PR1: Scheduling issues

All of our group members have quite busy schedules and it might be hard to find good times to work on the project together. Luckily, working at the same time isn't absolutely required and we can still make progress.

However, we want to be able to have short meetings related to planning and work distribution. We will make this happen by making the plans in the beginning of the sprint so that everyone can fit the meeting to their calendar.

Root cause: Hard to find fitting time for meetings.
Importance: 2
Avoidance: We will plan early in the beginning of every sprint.
Prevention: Everyone is committed to finding a place for the meeting.
Recovery: We will communicate in Slack if it's impossible to meet face-to-face.

3.4.2 PR2: Not starting early enough

This is more of an internal risk than anything else. We have noticed that we tend to start working on projects too late and the outcome is worse than it could have been.

We will try to avoid this by having a meeting in the very beginning of every sprint where we define the tasks for the next sprint and make other related plans.

Root cause: Being too lazy.
Importance: 4
Avoidance: A short meeting in the beginning of every sprint.
Prevention: Everyone's commitment to follow the plan.
Recovery: Magic during the last 24 hours.