Change Report

Approach to Change Management

Our change management system consists of the following stages: Change Identification, Cost and Risk evaluation, Planning, Implementation, Testing, Verification.
We used these stages as they can be applied to both changes in code and in our reports. We used GitHub issues to track change requests. We did this because we can easily track code changes tied to specific change requests, discuss changes as a team in the comments and have a single page to read to see all discussion and information regarding a change request. This is especially important as communicating changes is essential for successful software engineering. If two different developers are working on different changes without communicating, it is very possible that their changes will conflict with each other.
At any of the following stages, as a group we could reject the change. If we did this we spoke to the client about the impact on the requirements if there was one. Based on this discussion we would create a new, hopefully more suitable, change request.

Change Identification
As soon as we moved to the new code base, we re-read our client's initial brief and read Bugfree's formal list of requirements. This gave us an idea of changes we needed to make to fully meet our clients requirements. As a team we added every change we thought was necessary to the issues tracker. Then we went through each issue, and removed any duplicate or overlapping changes, or anything that we thought was out of the scope of the project. We were particularly careful to avoid feature creep[1] early on in the project when often team members are excited and have a lot of interesting ideas. However, many of these ideas can be more than what the client requested. Throughout the project we discussed new change requests by relating them to the requirements the same way.
Cost and Risk Evaluation
Although there is no monetary cost to our project, there is a time-to-develop cost associated with each change. As a group we decided on completion time estimates for each change request. Many software engineers agree it is very hard to estimate software development completion times[2], but in groups estimations can be better. We discussed whether the change would actually make the product better (does the client have conflicting requirements), whether it will fit into the existing architecture while keeping the code maintainable etc.
We also discussed any risks associated with the changes and added them to the risk register.
Planning
Planning usually consisted of a very short meeting where the team-member(s) who would be responsible for implementing the change discussed their plan with the technical lead and the project manager. They would evaluate if the plan had taken into account any associated risks and fully met the requirements the change was meant to meet etc.
Implementation
Code changes were implemented on a separate branch in version control so that they could be reviewed before being accepted into the master branch. This also means that if a change breaks something we can easily go back to an older version.
Report changes were also reviewed using the Google Drive suggestions feature, which lets users edit a document but keeps both the original and edited version until another user approves the change. There is also a backup/rollback feature.
Testing
Throughout implementation and after, we would run all existing and newly created unit tests. We would also spend a few minutes play-testing the game especially focusing on areas we changed. We asked ourselves are we confident that this behaves correctly. When assessing a code change, we also checked that any related code documentation and any report documentation was updated as well.
Verification
We referred back to our original change request on the issue tracker to decide whether the request had been fully satisfied. If the request was not fully satisfied we either pushed the issue back to the implementation phase, or closed the existing request, accepting the changes, and created a new change request for the rest of the request. We chose which of the above to use on a case-by-case basis depending on the request.

Summary and justification of changes to reports

Methods and Plans

See the updated document here
It was vital to ensure all team members regularly communicated with one another on what they are working on as due to the nature of this project at any one time there are multiple people working on the same software. We had different members taking ownership of the items i.e. the technical lead was responsible for the tests, the documentation lead was responsible for the documents and the secretary for the user manual. However, this did not necessarily mean they were expected to do all the work; they were responsible for making sure it was completed within the deadline and to a high standard.
For communication outside of meetings, we continued to primarily use Slack, but also Facebook Messenger. We found these tools to be a quick and effective way to communicate regarding meeting details, requirement enquiries, design decisions and progress updates. However, for assessment 3, we decided to create different channels on Slack depending on the topic of discussion. Our three main channels are: general, game development and deliverables. Our general channel is mainly used for organising meetings and for anything that does not relate to the other channels. The game development channel is for coding enquiries/updates whereas, the deliverables channel is for reports and anything else that needs to be written up. We decided to do this because previously all team members would receive notifications for all messages posted on Slack, whether or not that message was relevant to the them. This not only distracted team members but also resulted in team members getting frustrated and not checking all messages; resulting in important messages being left unread. Therefore, now if a team member wants to post a message on Slack they only post it on the relevant channel and only members involved in that channel will get the notification. Members are part of a certain channel depending on their role within the team (i.e. the technical team are all part of the game development channel).

At the start of assessment three, we discussed team roles and we identified that all team members were happy with the roles they had during assessment two. Thus, the roles have remained the same. Likewise, we kept the structure of our meetings the same by sticking to a 15 minute time limit and the 3 question format: What have you done since the last meeting, what you are planning on doing today and what obstacles are impeding your progress. We decided against changing this as we found having shorter meetings are more efficient; they are less time consuming and an effective way of keeping track of progress. However, due to this assessment having a shorter deadline we decided to have two face to face meetings a week instead of one. We felt this was necessary to ensure all team members are continuously working on the project and having more in person meetings will encourage the members to meet their individual deadlines by completing the work assigned to them.

In addition, we decided to use the ‘issue’ feature on Github more during this phase of the project. At the start of this assessment, as a team we converted all requirements into issues on Github, making sure to assign a team member to each issue and an estimated completion time (1 hour, 2 hours, 1 day or 2 days). This helped to keep track of everyone’s contributions and eliminated the risk of having two members working on the same issue. We decided to use this feature to prevent two people working on the same issue without the other knowing, this occurred a few times in assessment 2 and wasted considerable time. Doing this helped split the project into smaller, more manageable sections which made it easier to evaluate how much work was needed to meet the deadline.

Other than the changes mentioned above we kept our methods and tools mainly the same as our previous assessments. This was decided after a team discussion where we all agreed that we were familiar with our existing methods and were happy to continue.

New plan for Assessment 4 link to gantt-chart

Assessment	Tasks	Earliest Start Date	Latest Finish Date	Priority*	Dependencies
4	Choose another product	22/02/17	26/02/17	1	N/A
	Implement code	27/02/17	29/03/17	1	Understanding requirements

	Evaluation and Testing Report	05/04/17	02/05/17	2	Implementation
	Architecture and Traceability report	22/03/17	29/04/17	2	Implementation
	Project Review Report	22/03/17	02/05/17	2	Project Completion
	Assessed Presentation	03/05/17	07/05/17	1	Project Completion

* Higher priority tasks are given a smaller number and are only respective to other tasks within that assessment, not the whole project.

Risk Assessment and Mitigation

See the updated report here
We decided to continue using our original approach to risk assessment and mitigation. The reason for this is that the team members are now all familiar with the risk register and how to use it effectively, as we have used the same one since assessment one. If a risk occurs, we discuss it as team and refer to the risk register for the relevant mitigation technique. We prefer using a tabular format to display our risks as it makes them easier to read.

There were a few new risks that we identified at the start of assessment three after having chosen another team's product. The new risks (#7, #21 and #25) have been added to the updated risk register along with severity levels and mitigation techniques, and are summarised below:

- Our changes breaking existing code. To prevent this we used the existing tests to ensure we are not changing existing behaviour.
- Team members losing time trying to understand the code and architecture of the project we picked. To prevent this happening in the next assessment, in the plan we have set aside some time to read through the team's code and documentation thoroughly to ensure we keep on schedule.
- Slight requirement differences between our group and the group whose project we picked up. We decided to adapt the requirements of the team whose project we chose. We decided to do this as the game we chose was modelled to these specific requirements therefore, we thought it would be easier not to change/add additional requirements from our original set of requirements but to continue with the other team's requirements. We negotiated any requirements we were not sure about with the client face to face.

GUI Report

See the full report here.

We decided to use our chosen teams GUI report as a base for the updated report instead of our own from the previous assessment, because it is more relevant to the UI of their game than our own. Despite this, numerous changes had to be made to the report because a lot of the features that they claimed were implemented were missing. One of the ways we adapted the previous report was by changing reference style for the requirements. This system was the same one we used last assessment so all of the team was more familiar with this layout.

Many of the changes that our team made were just extensions to the GUI in order to complete the product, such as the main menu, countdown clock, market, gambling menu, auction, removing roboticons, random events and end game screen. However, there were several changes that we made in order to comply with our game design philosophies and changes that we believe are important for a game that will be played by a range of people, all with different levels of gaming experience.

Our justification for making these changes are to enhance the user experience by making features easier to interact with and information easier to obtain. This will benefit users of all experience levels. Also, by commenting these changes in our report, it allows other teams to see why our changes were made.

More detailed justifications on each change we made can be found in the updated full report.

Testing

Links: executable tests, executable test results, black box test scripts with results, all as .zip
We used both unit testing and black box testing as using a range of testing methods helps cover more use cases and catch more bugs. Different testing methods are also more suitable at testing different requirements. For example, its very simple to test that players and the market start with the correct amount of money in Unit Tests, but much easier to test GUI interactions with black box testing.

Unit Testing

Bugfree had written their own testing software, rather than using Unity Test Tools [3] or another library. We felt this was a poor decision as they had not tested their test suite, whereas existing libraries have unit tests, and many existing users who report bugs in them etc. We copied their existing tests into Unity Test Tools format, which uses NUnit. Throughout this process we also checked that there tests were correct, but some would pass even when they should not or were not testing what they said they were. Where this happened we wrote new tests to replace the broken ones. We also added new tests for code the existing tests did not cover, such as testing code actually returning expected values, when previously Bugfree were mostly testing if code threw exceptions when it was supposed to. This initial effort was worth the increased confidence in our product and easier regression testing. We are aware of the risks of testing. Incomplete or incorrect tests can give you a false sense of confidence in your software. This is why we chose to use more than one approach to testing. We also tried to reduce the likelihood of incorrect tests by checking each other's tests and checking tests were comprehensive by making this a mandatory part of code review. We made it clear that tests are as important as our code in order to be confident in our product. We felt unit tests were a vital part of our software engineering process, helping to spot unexpected side effects when making changes or refactoring existing code, and speeding up development and increasing the level of confidence in the correctness of our software with Test Driven Development [4].

Black Box Testing

As well as unit tests, we developed a set of black box tests throughout our development process. These test involve taking the role of a user and playing the game. However while doing so ensuring that certain criteria are met. For example, do buttons respond the way they should, does the GUI display the correct information etc. We encouraged users to write the test criteria into our black box test scripts document, so that they could be repeated after merging branches in version control, or by future teams. In the scripts we also include list possible edge and corner cases that tend to be bug hot-spots in software. These can include things like does opening a list of roboticons crash the game if the user does not own any roboticons. Black box tests were a useful addition to our software engineering process, because they give us a systematic way to test the behaviour of our software from our users point of view, this makes it much easier to link our tests to many of our requirements.

Bibliography

[1] MIT. “Feature Creep and the Importance of Iterations” miter.mit.edu. [Online]. Available: http://miter.mit.edu/articlefeature-creep-and-importance-iterations/ [Accessed: Feb. 20, 2017]
[2] innoarchitech. “Why Software Development Time Estimation Doesn't Work and Alternative Approaches” innoarchitech.com. [Online]. Available: http://www.innoarchitech.com/why-software-development-time-estimation-does-not-work-alternative-approaches/ [Accessed: Feb. 20, 2017]
[3] Unity. “Unity Test Tools - Asset Store” assetstore.unity3d.com. [Online]. Available: https://www.assetstore.unity3d.com/en/#!/content/13802 [Accessed: Feb. 20, 2017]
[4] Microsoft. “Guidelines for Test-Driven Development“ msdn.microsoft.com. [Online]. Available: https://msdn.microsoft.com/en-us/library/aa730844(v=vs.80).aspx [Accessed: Feb. 20, 2017]