Module Code
COM00008I

BSc, BEng and MEng Degree Examinations 2019–20

DEPARTMENT OF COMPUTER SCIENCE

Software Engineering Project (SEPR)

Open Group Assessment

Module	Software Engineering Project (SEPR)
Year	2019/20
Assessment	3
Team	The Dicy Cat
Members	Michele Imbriani Daniel Yates Luke Taylor Isaac Albiston Martha Cartwright Riju De Sean Corrigan
Deliverable	Change Report

CHANGE REPORT

The team’s formal approach to change of management was organised and efficient. During the time prior to the projects’ presentation day, the team discussed and took note of the **key elements to look for/look out for** in other teams’ works. This allowed us to have a very clear and well-defined idea of what direction our team wanted to proceed in for Assessment 3. Multiple areas were discussed during this process; nevertheless, the ones that we assessed to be of paramount significance -and which, therefore, had the most influence towards the final decision- were: 1) similarity between our game and the new game; 2) clearness and understandability of code and documentation; 3) availability of the other team to answer queries; 4) clearness in organisation and explanation of the deliverables. After the decision was made, we immediately identified the **areas of the new project that needed attention** the most, thus assigning them higher priority over the others i.e. code, game-map/art style and requirement reports. Because the use of version control platform GitHub proved to be highly effective during previous assessments, along with all team members being acquainted with it by now, the next straightforward step was to create a new repository containing the new project’s code, creating separate branches where necessary.

With regards to **the code**, we employed Lehman's Laws of Increasing Complexity [1], Continuing Change [1] and Continuing Growth [1] in order to have a better understanding of the complexity and magnitude of the work that laid ahead of us. Spending a considerable amount of time and effort in the project selection process revealed to be a wise choice: the code was overall very well organised and documented, which immensely helped us laying the foundation for a successful and strong result. Because of this, it was not necessary to make any significant changes to the existing code or features in place, but only needed to make very few corrective changes, and were also able to implement assessment 3’s features with relative ease. Furthermore, when making additive changes, we were careful to refer back to the original architecture documentation -and, in some cases, contacting the project’s authors directly- to make sure we added the new features in the best way possible. Finally, we decided to stick with the previous groups' commenting and coding practices to keep things clear and to avoid creating unnecessary confusion.

With regards to the **change management of deliverables**, we recognized that the transition between our team’s previous requirements and the new ones can be interpreted as a P-program, as defined by Lehman [2]. Implementing some of the features elicited by the previous team’s requirements was a straightforward and clear problem in specification, but because of the differences between the two projects, it turned out to be more challenging and more inaccurate in the real-world context. For example, the newly picked project had a level-based game style (UR_select_level), which was completely new to the team. Another straightforward example is the requirement (and relative feature’s implementation) *FR_save_quit*, which was something that we had not implemented in our original project, and which required more effort to wrap our heads around than predicted. Fortunately, the UML architectural framework provided by the other team helped us getting a better understanding of how to implement requirements and features in the safest way possible.

When approaching the new project’s **Risks Register** [3], we recognized that the previous team did follow the same risk elicitation and mitigation method employed by us. What changed, however, is the use of a separate **Risk Tracker** [4] for keeping track of the modifications made to the risks table. This is an innovative idea that surely aids the immediate understanding of what has changed, but it does not follow -or if it does, it was not explicitly specified in the Risk Assessment and Mitigation document- any specific method from any textbook. Despite understanding and respecting this choice, our team agreed that because of this feature being useful but not necessary, thus omittable. Apart from this, no further changes were made to the risk tables other than the addition of risks which had not been covered yet, using our assessment 2’s risks table as reference. Our updated Risks table can be found on our team’s website [5].

TESTING REPORT

For this assessment, we decided to stick with **two types of testing**: White-box testing and Requirements testing. The purpose of the former is to thoroughly assess and examine the code's syntax, semantics, inner workings and structure. The latter was introduced in order to provide a higher-level layer of testing that is lacking with the white-box testing method, and whose aim is to test whether the implementation of the features elicited in the requirements was fully achieved or not. The **reason for choosing strictly White-box testing** over a combination of White- and Black-box testing was based on the decision that, unlike in the past assessment in which a sub-team was assigned strictly to testing, every team member would be involved in the test development process, thus eliminating the possibility of black-box testing in toto.

We decided to research and experiment with different testing methodologies that would suit better the purposes and scope of our project. Among all possible methods we considered, **Test Driven Development** seemed like an ideal method to adopt for our project. As explained in the paper by Janzen and Saiedian [6], Test Driven Development is an agile testing strategy that "requires writing automated tests prior to developing functional code in small, rapid iterations" [6]. Thus, TDD is a perfect fit for our project because not only does it provide a quick, precise and iterative approach to writing tests which is perfectly in line with the Scrum agile development method used by us, but it also encourages quick refactoring of code that allows the team to improve the code without causing major breaks or bugs whilst providing the necessary flexibility to adapt to changes in requirements.

Another new testing approach we implemented in this assessment was the use of -or, better, the formalisation of- **Peer testing**. This involves reviewing the code of other team members to identify areas for improvement. Despite everyone in our team already having employed this method in multiple instances throughout the course of the last assessment, we decided to dive deeper into it and research appropriate literature to support our technique. This allowed us to formalise a practice that was already informally in use by all team members, providing us with a more systematic and technical method for Peer reviewing. We consequently found evidence that Peer testing improves the quality of a software product and also encourages steady progress "since the students need to show their work to peers before it is actually due for submission, they are required to begin working on it much earlier" [7].

We decided to stick with our tool of choice for automated unit testing adopted in assessment 2: JUnit. The advantage of automated testing in the context of the current assessment is that tests are reusable and are therefore more reliable and consistent than manual approaches to testing. [6] On top of that, since the whole team is involved in producing tests as well as working on the implementation of the features for assessment 3, employing automated unit testing allows the team to reuse tests and improve time efficiency.

Our testing evidence, along with traceability matrix and tests table, can be found on our team's website [8]

METHODS AND PLANS REPORT

For this assessment, the **Agile Scrum** development method was employed again. This was optimal due to the short timeline imposed by the assessment deadline, hence iterative software development encouraged by Agile was desired.

One of the **most impactful changes** that took place at the beginning of this assessment concerned the **team organisation**: as agreed at the very beginning of assessment 1, the establishment of highly flexible roles and the willingness of the team members to work on all areas of the project development allowed the team member Luke to take on the role of Project Manager instead of Michele without experiencing any major drawback or issue. This change turned out to be a winning move on both ends. **On one hand**, the new Project Manager showed a high organisational ability both team- and tasks-wise, coupled with an equally high sense of leadership, commitment and enthusiasm towards the role, which undoubtedly provided the team with revived energies for tackling the new assessment. On top of that, his strong coding skills were another suggestion that this team management change was a right fit. **On the other hand**, downgrading Michele from the role of PM not only did it allow him to gain a more down-level, hands-on knowledge about the overall game which was somewhat lacking due to PM duties, but it also allowed him to focus more of his efforts into the supervision, review and production of all the deliverables, which, seen the positive results of the past assessments, is something that he has been consistently performing well on. Finally, having understood the importance of the management role, the team, and especially the new PM in charge, relied on the information contained in the textbook Peopleware [9]. Despite the book not being software-development focused management, and despite almost half of the book dealing with how to deal with underperforming members or teams (which, fortunately, is not our case), this literature proved to be an essential reading.

When it comes to tasks division, the team made treasure of the experiences made during the previous assessments: we recognized that each component in our team has strengths and weaknesses in different areas which do not necessarily overlap with other members. For this reason, we agreed upon splitting the tasks that needed to be done according to each member's past assessments' experience. Without loss of flexibility, **four sub-teams** were formed: implementation of features, documentation, minigame and game map. On top of this, using the Scrum method proved to be highly effective in letting team members interchange between the four sub-teams as needed, whilst keeping track of the progress made and the tasks that needed more attention than others.

Concerning the **tools employed** in this assessment, some new options were proposed and considered throughout the course of this assessment, but overall **no major change is there to report**. One of the tools that was used initially but abandoned soon after -and which we discovered through the Summary of Tools and Method Changes [10] deliverable from NPstudio- is TeamGantt, an extension of Trello which allows to easily and quickly organize and visualise tasks in a Gantt Chart fashion. Despite this feature sounding very appealing, we agreed that it was not suitable for our goals and applications: the way our team decided to use Trello, which proved to be extremely useful and efficient in the past, is an informal tracker for tasks and their relative progress status throughout the Scrum sprints, which clearly are not representable with a GanttChart. The same reasoning applies to another tool which we considered: GitKraken Glo Boards. This is a tool very similar to Trello, but which can by synchronised with GitHub, our version control platform of choice. But once again, despite this feature being very tempting and surely an immensely useful software development tool, we agreed on it being of little use for the scope of our project.

Our updated Plan for assessment 4, along with relative Gantt Chart, can be found on our team's website [11].

[1] Lehman, M. M. (1980). "On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle". Journal of Systems and Software. 1: 213–221. doi:10.1016/0164-1212(79)90022-0.

[2] Lehman, Meir M. (1980). "Programs, Life Cycles, and Laws of Software Evolution". Proc. IEEE. 68 (9): 1060–1076. doi:10.1109/proc.1980.11805.

[3] NPstudios' Risk Register: https://npstudios.github.io/files/Updated_Risk_Register.pdf

[4] NPstudios' Risk Tracker: https://npstudios.github.io/files/Updated_Risk_Tracker.pdf

[5] DicyCat Updated Risk Register: https://dicycat.github.io/files/UpdatedRiskRegister.pdf

[6] D. Janzen, H. Saiedian, "Test-driven development concepts, taxonomy, and future direction", Computer (Volume: 38 , Issue: 9 , Sept. 2005), 2005, [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1510569. [Accessed: 3 December 2018]

[7] N. Clark, "Peer testing in Software Engineering Projects", ACE '04 Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30 - Pages 41-48, 2004, [Online]. Available: https://dl.acm.org/citation.cfm?id=979974. [Accessed 3 December 2018].

[8] DicyCat's Website: https://dicycat.github.io/assessments/

[9] DeMarco, Tom., and Lister, Timothy R. *Peopleware : Productive Projects and Teams*. 2nd ed. New York: Dorset House Pub., 1999. Print.

[10] Summary of Tools and Method Changes https://npstudios.github.io/files/Updates2.pdf

[11] DicyCat Plans for assessment 4: https://dicycat.github.io/files/PlanForAssessment4.pdf