SEM: reflection on the learning process

Group 66

Game: BustaMove

Contributors:

- Antony Löbker
- Omar Hommos
- Jeroen Overman
- Vito Kortbeek

As a start, members of this group are MSc students who are taking the course as a homologation (mandatory for 3 members, optional for 1 member) as a part of their master's degree. Background of the members is Electrical Engineering for 3 members, Computer Science for 1 member. Three of the members don't have a previous experience in Object Oriented programming, but rather a background in C or C++ and in Embedded Systems programming.

The first submission was relatively quite good, especially when taking into consideration the background of the members and the fact that they started the course several days after their peer members. Getting acquainted with OOP was easy. However, SE practices weren't followed thoroughly as students had no previous exposure to that, and were used to procedural programming.

With the start of the lectures and the lab, more experience got acquired. Some aspects of OOP were somehow weird for 1 member (Especially that it has a lot of overhead, in comparison with programming microcontrollers with the goal of keeping the software size as small as possible).

Getting to notice the mistakes done in the initial version happened first during the UML lab, as doing a UML for some parts of the game made the team feel where they introduced unnecessary complexity, and where things must be done in a better hierarchical organization. With the later two labs, the importance of design patterns was well understood. However, being forced to use a specific number of DPs out of specified ones was somehow annoying.

Evaluating a peer team code as a part of one assignment was great. It helped gain more experience regarding other's code. We highly suggest this gets repeated again in another assignment. Things learned from this part of the assignment included getting familiar with other tools, understanding different code style conventions, familiarity with other design patterns

(whether it was taught in the lecture but not applied by our team, or other DPs applied by the other team).

Using the InCode to evaluate the code was a quite good experience too. It helped detect some problems (some we knew, some we didn't), and gave us a good chance to fix it. More importantly, it showed how the code can get messy and tangled with the sudden increase of its size without previous planning of that increase. The importance of software engineering design principles and method showed a significant advantage with this i.e. the importance of how design affects extendability.

As for the testing, being forced to have a high line coverage made us realize some mistakes in the code e.g. high staticness, wrong hierarchies, etc.. Working with testing was a good experience, considering that three members have no previous background in this whatsoever.

In general, team-work experience was a very valued thing to learn. Knowing how to properly interact to reach a common goal, and how to work on code quality evolution as a team was especially valuable. Getting familiar with SCRUM was quite a good process. The improvement in the planning and retrospectives of the SCRUM process was felt week after week (Especially with the time estimations becoming more accurate eventually, thus improving task splitting).

In conclusion, the difference between the first and the final version:

- MUCH more expandable and testable.
- Much more modular.
- Easier to expand and add new features.
- Can be ported easily to other games e.g. if another game make use of a graph to store elements and a relation between them, this can be a good starting point.

As a feedback on the course:

- It is hard to do that, but can students get introduced to the names of the design patterns that will be taught in the very beginning of the course? Some problems were faced at a point where a specific design pattern was a good solution. However, as students were not familiar with it from the lectures, they didn't get to know its existence and use it. Knowing the possibility of DPs that will be taught might encourage students to start exploring earlier, and apply specific DPs even before getting to know them in the lecture.
- The interaction in the course is quite nice. From using Slack to having a direct informal contact with the TA; this interaction method distinguished this course from other courses. The TA was quite nice, always ready to explain stuff. His feedback was very important. One of the great aspects on having a TA-team mentoring is that his comments were on-point, and were based on his observation of the code evolution.

- Parts of the exam involve memorizing more than understanding i.e. the first part of the midterm. As software engineers, we learn and go through a process. This process will be applied and will become a good habit with time. However, being asked to label and name parts of some process or cycles didn't make a lot of sense. Furthermore, MCQs can be a little bit more specific in some cases, which will help students select the correct answer with less hesitation and fear.
- Feedback on the grading process:
    - Having a separate midterm and final is awesome.
    - some team members have the suggestion of increasing the lab grade to 60%, while keeping an ever split between the midterm and final. Furthermore, as the lab and practical experience reflects the progress and the learned software engineering trains more than an examination, it might be better to just require a passing grade for the whole subject, and not for the (exams && labs) altogether.