

COMP47480 Contemporary Software Development

Lab Journal

Sukrat Kashyap (14200092)

BSc. (Hons.) in Computer Science



UCD School of Computer Science
University College Dublin

April 27, 2018

Table of Contents

1	Extreme Programming	2
1.1	Work Done	2
1.2	Reflections	2
2	Modelling with UML	5
2.1	Work Done	5
2.2	Reflections	5

Practical 1: Extreme Programming

In this practical, we role played and tried to understand the iterative development also known as agile development in Extreme programming.

1.1 Work Done

We were a group of 5 people. Our task was to build a fridge using the Extreme programming methodology. So, at the start, we divided our group into 2 consumers and 3 developers. I was one of the developers in the Iteration 1. Consumers were then supposed to create stories telling about the features they want in the fridge. These stories by the consumers were given to us to give the expected time to finish the feature. We developers then estimated the time required for the stories and gave the cards back to the consumers. Some stories were not clear like "It has two-sided door". So, we asked the consumers whether they want the doors for both the compartments or one big two-sided door for both the compartments. After clarifying this detail, we drew a small picture of the same to make it clear. After giving the cards back we gave our work time of 3 minutes for the iteration. The cards were given back to us and us three developers one by one took stories and implemented them.

In similar fashion, we did the second iteration. Only this time the roles were switched. So, then we wrote the features on top of the fridge that was already being created. The developers clarified some details such as how big TV they want. The estimates of the stories and their total work effort were given. We then picked all the stories as the estimated time of the stories were completely fitting their effort time. They then took the stories one by one and build the device on the previously implemented fridge.

After, the 2 iterations we retrospect on the whole process. The customers were happy with the product. Developers had few difficulties implementing as some of the requests were bizarre but doable. Queries were solved satisfactorily but most of the decision was given to developers as what they might seem fit.

1.2 Reflections

Extreme Programming is one of many software development methodologies which aims to improve software quality and responsiveness to changing customer requirements. It is a type of agile software development which has both incremental and iterative properties. These properties tackle the varying project requirements (One of the major problems in software development).

In the practical, we used Extreme programming methodology to develop our fridge. Only 2 iterations were performed which was enough to give us an insight into the development process. The difference between two properties namely *Incremental* and *Iterative* was also clear. In simple terms, *Incremental* was adding new features to our fridge and *Iterative* was adding these new

features repetitively. It also helped us to understand the developers and consumers mindset. I felt that it helped me understand the 4 main agile manifestos in practice:

Individuals and interaction over process and tools EP felt like a program which decoupled two entities (business and development) in software development and created a link between the two.

Working software over comprehensive documentation While developing fridge there was no guideline of documenting or even thinking and caring about future needs but it strongly withheld the idea of testing each feature before saying its complete.

Customer collaboration over contract negotiation The developers collaborated with the consumer to understand the feature of the fridge before estimating their time and communicating when implementing the same.

Responding to change over following a plan Few modifications of the original feature was done to accommodate new feature in the second iteration.

One of the other advantages that I felt was that the process was much simpler to understand and implement. Unlike other Agile practices such as Scrum. Scrum seems a bit complicated as it contains various roles such as Product owner, scrum master, team members etc. and contains multiple artefacts such as product backlog, sprint backlog, burn down charts, Taskboard etc. Whereas, when it comes to EP the process is much simpler having only 2 roles that are the developers and consumers with the artefacts being the story cards. In all my internships scrum was the main methodology of development. It seems to have fine detail about each role and seems to be a bit less flexible. Whereas the EP abstracts the management of each role. Giving the flexibility to the developers and consumers to manage their internal affairs on their own. EP seems to look like a collaboration and interaction process between the developers and consumers. While saying this, EP does have guidelines which the 2 entities must follow for better throughput [1]. For e.g.

- For developers
 - Pair programming
 - Testing first approach also known as (TDD Test driven development)
- For consumers
 - Stories must be a feature that the consumers with no implementation detail
 - Stories chosen to be developed must not be more than the promised effort number given by the developers

The incremental property of Agile practices, in general, gives the business side of the company to evaluate their product in the market after every release which helps to reduce the cost. Unlike the Waterfall process where the defect is usually found in later stages, making the process revisit analysis, design and implementation part. Whereas in an agile process, the defects are found early, giving the opportunity to the business side to re-evaluate their product. Which usually involves in a spin-off of the already released product and adding new features to counter-attack the problems. E.g. we had an ice and iced water dispenser in the fridge and if it didn't seem to work to attract customers, in iteration 2 we added ice cream maker to attract the customers.

In the end, it felt Extreme programming was a nice way to development. But still, it lacked stronger and finer guidelines and rules when it came to a bigger team. We were a group of 5. So, it seemed easy to follow and implement the ideology. But it seems to be naive when it comes to

bigger teams on the much bigger project [2]. Since its iterative, it seems harder to create software which has deadlines. One of the problems that I realize in Agile, in general, is that when the developers start working on an iteration the stories are locked and not allowed by the consumers to change. So, a major defect or bug in the system which needs immediate care during the iteration is harder to include. Also, most of bugs or defect are minor and doesn't need many changes but some of them are major and it becomes harder to estimate the time to fix. The most problematic part of the EP is the estimation of stories, in EP the developers estimation is somewhat vague and is not always true. EP says to finish the story for the time being estimated. But, it doesn't happen. Estimated time is mostly underestimations and it is due to different developers have different speed in development. Even though other agile methodologies sometimes use a vague numbering system for estimation called story points. If the stories are not finished then they are moved to the next iteration/sprint. A team's number of average story points achieved in a sprint in the past and their current estimation is used as a measure to estimate the total work effort of the team. This seems a better approach than asking the developers their effort time for an iteration. Overall, EP is a more engineering focused methodology which seems to work best on small teams.

Practical 2: Modelling with UML

This practical we learn modelling the software using use-case model, domain model and interaction model.

2.1 Work Done

In this practical, we were given to create Use-case model, Domain model and Interaction model of a library system. The library had books and journals which could be borrowed by the members of the library (Students and Staff). There was a restriction on the number of items each member could borrow. Journals could only be borrowed by the Staff members. There were two types of loan for the books namely short-term (4 hours) and long-term (4 weeks). We had to represent the system in 3 different model representation. We first created use-case model for the library system which was done by finding different actors (student, staff, and member) and use-cases such as borrow journal, check availability and common use-cases such as borrow book, return, and display. After the use-case model, we picked out nouns suitable for making classes like Student, Member, Staff, Copy, Book, Journal and created links between them. We then created an interaction diagram which showed us scenarios of communication between the classes and had to add a new booking system to our domain model. We then were told to add a new requirement of notifying the members if they had the book or journal for too long. To which we added a booking system which notified the member when the booked in all of our models.

2.2 Reflections

We started modelling our library system by first created use-case model based on the description. For which we had to find out the actors and use-cases. We thought of actors as the staff and students. Our use cases were borrowed, return, check availability and display. Then we tried to reduce the functionalities by excluding and include generalization. Student and staff were generalized using the actor member. This reduction of both the actors to members was because they shared mostly all the common functionalities except borrowing journals. All the use cases were included in the display because they shared functionality. An actor performs display then borrows/return and then displays new changes in the system. Borrow journal use-case was given separately to staff as only they are allowed to borrow journal. A common return was used instead of return book and return journal as by using simple logic as to when a student cannot borrow journal, it won't be able to return it either. Hence common return for both made sense as they can only return what they borrow. This form of representation seemed quite useful as it was able to showcase various parts of our system in more readable format. A paragraph of description is hard to understand in one go. This form of modelling enabled to look at the system in much user-friendlier format. Regardless, it failed to showcase much tinier important detail such as students are only allowed to have 3 books and the types of loans of the book available.

The second form of modelling which was done using the description and the use case model was

a more of a class diagram that would have been implemented when actually writing the code. It was done by first identifying the nouns. In our case, we thought of Student, Staff, Member, Book, Copy, and journal as our nouns. The second step involved finding the relationship between the same. Student and Staff inherited from the members. This was thought as library system should just care about the member and the limit imposed on them should be handled by their concrete classes student and staff. The reason Copy was created as the book could have multiple copies with a different type of loan but all the other detail would remain the same. Hence N:1 relationship was identified in the Copy vs Book. Since journal didn't have any copies so the copied class was omitted from the journal. The borrow and relationship were straightforward from our use-case model. Comparing the two models above, we see use-case model tends to just identify the actors and what they did irrespective of how they did. Whereas, domain model representation adds a technical aspect of the system. This could be easily seen from the attributes corresponding to each class. Both representations are quite similar to domain model being a bit more technical. Domain model could also show them the fine detail of the limit on each student and staff by assigning a constant value of 3 and 12 respectively to the limit attribute they both shared. We decided to skip such fine details. But it gives us insight into the level of detailing without making the model too complex can be achieved. Although, it seems like a better option than domain model. I feel Domain model has the potential to become too complex real quickly than use-case model.

The third form of modelling was Interaction diagram which tries to picture the exact interaction between the general ideas but more in sequential form. We took member, booking system and item to show the interaction. It shows the sequence of getting items from booking system which in turn called each item and their info. It returned a list of items and their status. Borrow item can be issued later passing the exact item one wants, the result was the confirmation. Notify was included as this was the third requirement which would be done timely until the user returns the exact item. Our sequence diagram was not as perfect as it should be. But I believe it did represent the sequence in the steps for e.g. return can only be done if borrow for that item is issued and notification will be done until the item is returned. Notification step was also added to other diagrams as an addition which didn't seem to be much of a hassle.

After building all the three models, one can see that the use-case model represented the system in a vague manner missing all kinds of detail and giving a very high-level overview. The domain model gave simple and fine details and also somewhat the implementation of the same whereas sequence diagram gave those implementations and relation in the domain model a sequence in which to be performed. But when it comes to usage of these above three models in real life, I believe it is not as useful. I believe these models were highly required and recommended in the past because of there were none to very few development tools and required a great deal of work to manage and learn about the new system which was already implemented. In today's world, we have so many advanced development tools such as IDE, debugger etc. that it has very much diminished the use of modelling. Also, the changes in the project are very rapid making models before coding redundant. A great example to show this case would be finding the implementations of an abstract class. Before IDE's finding a class in a big system which implements certain class was extraordinarily time-consuming. Hence, the models such as UML helped to figure them out faster. But in today's computing finding the implementation is a task of a mere single click. With that being said, simple to represent the whole system still exists in form of vague pictorial drawing rather than following a standard.

Bibliography

- [1] Lee Copeland. Extreme programming, Dec 2001.
- [2] Matthias M Müller and Walter F Tichy. Case study: extreme programming in a university environment. In *Proceedings of the 23rd international conference on Software engineering*, pages 537–544. IEEE Computer Society, 2001.