2/28/2018

COMP47480 – Contemporary Software Design

Learning Journal

Orla Cullen
14708689

Table of Contents

Comp 47480 Learning Journal – Assignment 1

1. Reflection and Account on Team

First, we split our team into the customer and the developer groups. Myself and Jia were the customers and Carl and Neha were the developers in the first iteration. Myself and Jia began to specify our user stories. Whilst we defined the user stories the developers took each card and estimated how long it would take to implement. In hindsight I feel that some of the user stories we defined could have been improved upon. The implementation of the user stories we left down to the developer which I think is a requirement of the process. However, for some features and user stories it may be advantageous to seek clarification on the location of the features within the fridge. I realised this after we defined that the fridge should have a light that comes on when the door opens. The developer implemented the light at the bottom of the fridge, but it may have been better to be located closer to eye level. Overall in the first phase all the stories that the customers asked for were implemented. In the next iteration we switched roles. In this iteration the customers decided to update small features for instance having a second door for the freezer area and a second light so there would be one in the fridge and one in the freezer. Again, on this iteration it was noted that the design that the customers had compared to the developer differed slightly. In that the developers implemented an internal freezer door, but the customer wanted an external freezer door. This highlights how important proper specification is on the part of the customer but also the onus is on the developer to think about the specification and all the issues, lack of clarity and different ideas each party can have on the design concept. Overall both the customers and developers were happy with the overall product made, the stories presented and the deadlines for the project were met. However, the most important aspects of the planning game are simplicity clarity and communication and as a group we could have designed a better product had communication been more open and more questions asked. It is also worth noting that this product was a simple design in that everyone knows what a fridge is and what it does however when you apply this to software development the product will not always be clear and more input from the customer would be needed to guide the development of the product.

2. Reflection on my Learning on the Planning Game

The planning game is a technique that can be used in the extreme programming model and is a significant element in mapping out the design of the product. It allows customers and developers to interact and communicate with each other in a way that is advantageous to the development process by allowing customers to provide an outline of the features they require and developers to estimate how long each part of the process will take. It accepts the fact that the design process is not finite, and features and functionalities are subject to

change. Subsequently, the idea is that the customer can prioritise functionalities and the developer can then correspond with the customers on how long each aspect will take and whether each is possible given the order the customer wants to have the design completed. In this way this process is iterative, and the design can start as a skeleton and very basic concept after each repetition of the cycle the progress and specification of the functionalities and features can be enhanced. This process iterates through many stage many times until the required product has been achieved.

The planning game plays an essential role in the creation of common goals by communicating ideas. It essential breaks the process into two main stages, the design stage and the iterative stage. As I observed the planning game, it became clearer that the design phase was the part of the planning game that was based from a collaborative point of view, to understand what the essential requirements of the project are, to ensure that you as a developer have an understanding the requests of the customer. This can be achieved by listening to the customers' requests and being informative and communicative about what is achievable in the timeframe of the iteration which usually lasts 2-3 weeks. An important element in this process is to ensure the customer specifies all the user stories. The developer only estimates the timeframe of each user story or reject user stories that are overly complex so the customer can split the story into smaller more manageable stories or ask for clarification on some aspects of the story. It is vital that the user stories consist of 1 clear sentence to eliminate confusion and complexity. During this the stories are prioritising by the customer and the developer and the customer commits to the design until the next meeting.

The iterative stage is the process that just involves the developer. This is the stage where the developer goes away for the build phase of the project and implements the stories that the customer has specified. They don't diverge from the customers design they implement the design as given to them. This may involve assigning tasks to persons in a team of programmers. When the build is achieved it is released and the next iteration occurs where aspects can be redesigned, new stories can be added and stories that aren't reaching the mark can be deleted.

This planning process is really all about cycles of development and communication and a willingness to keep the design flexible so changes in requirements can be embraced by adapting the design at the end of every cycle. Due to it being a part of the extreme programming model in which testing is seen as imperative to a project so it is one of the first things done errors can also be addresses in a timelier manner. The downfall for me in this model is how many iterations must be completed before the finished product is produced. It seems like that the customer could redefine and create more user stories to implement in the project indefinitely.

Comp 47480 Learning Journal – Assignment 2

1. Reflection and Account on Team

During the practical our team first started the assignment by addressing the use case model this was an important part to defining what the library would do. Whilst reading through the instruction we picked out that there were 2 types of member that would be able to access the system. The student and staff would be a generalisation of the model whilst every member could borrow and return books only staff could borrow journals and to return either then there is a dependency that a book would be borrowed in the first place. I found this one the most difficult as I wanted to be able to define more things the library could do. I found it difficult to hone in on just the basics. We then moved forward to the class diagram which we have used more and used to show that the library consisted of items which were either of type book or journal and members that were either of type student or staff in this diagram we were able to define attributes that each class might have for instance a time period due to the fact that we could take out a book for 4 weeks and others were on short term loan. This for the team was relatively straightforward. Subsequently the sequence diagram could be addressed. This diagram was to show an action that might happen within a class on a timeline. The first sequence we decided upon was to borrow an item it the first sequence the system checks that the member has not reached their limit of items they can borrow. The second sequence that we had to design was to send an alert to the user when their book was out of time and needed to be renewed. This functionality required an interface to send the requests to the correct class (as seen in the artefact). In this the system runs a query on all the items in the library and checks the time borrowed against the last borrowed date if it is over the time allowed it then finds the user who has the book out and alerts the member.

2. Reflection on my Learning on UML

After attending the lectures and the practical given about UML, it has highlighted to me my knowledge of UML and the its different models has been quite limited to this point. Previously, I had used UML to depict class diagrams and had little knowledge of the other diagrams that UML encompassed. I was aware that UML could be useful tool in many industries in providing a methodology of how to approach constructing a mock-up of the system that team is implementing and providing a tangible visualization of how the system will work or what the system design will do to senior management that may not understand the system fully if the team was to describe the system by using technical jargon alone. UML is a graphical language used to model systems using relationships of the components and the dependencies between these components. I have learned that while I have only really used the class diagram. The unified modelling

language consists of use case diagrams, class diagrams, sequence diagrams, collaboration diagrams, state chart diagrams, activity diagrams, component diagrams, deployment diagrams. The use case diagram is probably the simplest form of diagram as it just represents what the system can do. It doesn't provide implementation however it does consider dependency of features. It can help in deciding what features are necessary in the system. The domain model /class model is one that I have previously used this I find provides a useful first mock-up of the system by conceptualizing the essential classes and features. It makes use of relationship between components which can be described in tree ways association aggregation and generalization A generalization occurs when there is a class that inherits functionality from the super class. An association is when there is some affiliation between classes and aggregation is when a class belongs to a collection. Arrows provide and idea of the direction of the association and we can provide a way of showing the number of possible instances of the class by providing multiplicities like zero or one instance by using 0.1 notation. The sequence diagram shows how the model interacts a with the classes in terms of the operations. These are read top to bottom not left to right as they are depicted like a type of timeline. Collaboration diagrams also provide a representation of interactions however it is focuses mainly on the role of the objects. I feel this diagram is more useful as an interaction diagram and I would lean towards using this as I feel it's closer to the class/domain models that I'm comfortable illustrating, so this feels like a good way of providing some modelling of interactions.

Personally, I find some of the UML diagrams simplistic and find it difficult to depict some of the diagrams because I tend to lean towards putting too much of the implementation and complexity into the diagrams where simplicity is necessary. I would much prefer to mock up the bigger picture of the system than to simplify it, to the point that I feel the diagrams can become too simplistic and insignificant to warrant illustration. Although UML is not continually utilized in industry It remains a useful tool where the abstraction of the model is necessary to maintain common system goals and design concepts between programmers where systems can become complex and difficult to understand. It also provides a common language that the everyday person can understand due to its graphical nature. I feel through the course of this practical I have gained a better understanding of UML for demonstrating the model of a system.

Seminar 1- IBM – Watson Health

This seminar was given by Paddy Fagan who is the chief architecture Watson care manager development. Watson Heath is a division of IBM who operate in 170 + countries and generate a revenue of $79 billion. This software cemetery around data analytics and cognitive insights dedicated to the health domain. Due to the sector there are certain legal requirement for the development process for instance one team develops and the other ensures the finalization of the software and its customer readiness. It is important to note that the way in which Paul and his team work within Watson health may not be wholly indictive of how IBM work although some of the process will be similar. If a new developer joins the team the normal time it takes to get setup with the correct files and IDEs would be anything from 5-10 days. They require developers to use certain tools as I ensure all the developers in the team have a similar setup and the files and paths needed have been bundled together to allow easier installation when you use Eclipse as your IDE.

Software Methodology:

The software is based from an agile development model where it supports collaboration and the evolvement in software development b being adaptive and flexible to rapid and radical changes in requirements. They pride themselves on the belief that software development involves many aspects like sales support and operations thinking of software and it should be a collaboration among all these different areas. Whilst the engineering aspect also should be considered in many different forms from project management, design, development, testing, pricing support and operations each one of these plays a vital role in the production of new technology /software. This creates a collaborative environment between all aspects of the software development and a central vision of how the user's software should work. They make use of sprints and operate in 2*2 week blocks the first block involves the commit and acceptance of the new code and the second centres on the release acceptance and the SRE validation stage before the process in transferred to a different team that will focus on the finalization of the code and deploy the build for customer readiness. After the acceptance stage the branch will normal be forked this allows there to be many versions on the go at a time and if there is a major issue with a branch then it can be easily overwritten by a newer branch. It makes use of Storyboards, user identifiable features, iterations of the design process and sprints. Speed is paramount in the design process and they don't appear to have a set way of coming up with the design where developers can make use of documentation wikis post its mock ups and presentations.

Use of Modelling:

The model of the system architecture is achieved by using IBM Rational Software Architect Designer. It incorporates the unified modelling language (UML) in the designing of software applications. It is a program built on eclipse, so this is another reason that the company likes to use Eclipse as its IDE. This tool allows access to cloud services and generates UML and

aids in maintaining control of the architecture. The functionality of this tool is easily extendable with plugins

Testing:

Watson Health do not use test driven development. This means that they don't require their developers to write their test cases before writing the software. However, they do require the use of JUnit as a testing suite. Functional verification tests are implemented which evaluate the logic of the feature design. It ensures that each function does what is intended. In comparison, system verification tests verify and validates the software meets the specifications of the software development process. Both types of tests are essential to the integrity of the software design however functional test would need to be passed in advance of running the system verification tests to achieve accurate results.

Refactoring:

Refactoring is done in a tiered process and it's very much a part of the company process. In general developers are encouraged to refactor as they go if deadlines of the sprint can still be easily maintained. However, if the necessary changes are extensive and it is thought that it will take a significant amount of time it is common for to leave the refactoring and ensure that it is added as a task to be completed in the next sprint. Radical refactoring only really takes place when the software has reach a point where its functionality needs to be extended because it no longer meets the requirements of the project. This type of refactoring would normally be delegated to the team by the team leader or someone in a senior technical role who has knowledge of features that may be required down the road.

Software Quality:

The company ensure the quality of the software developed by making use of architectural description during the design process. This allows for the analysis of risk during the development phase if there is some risk involved it is identified early and the risk can be accepted for declined after which alternatives can be investigated. In a nutshell this allows for the expression and understanding of the risk early on. Pair programming in the opinion of the speaker does not work all that well. So, it is not commonly done as it doesn't normally produce great result it is much more fruitful to work together where both programmers have access to a keyboard rather than taking coding in turns. Code reviews are used by Watson Health usually Paul is responsible for the code reviews of his team and where problems occur with the code he will sift through the commits to identify the individual responsible for a new code smells or bad code. SonarQube is used extensively by Watson Health it provides a continuous analysis of the code quality and offers reporting on duplicate code, how well the test cases cover the code aka code coverage, it can assist in detecting bugs and security issues and give an indication on the complexity of the code. This tool can also be integrated as a plugin with many IDEs. SonarQube is also a great tool for watermarking and has an extensive range on statistical report which can be activate if required.

Comp 47480 Learning Journal – Assignment 3

1. Reflection on my Learning on the Test-Driven Development.

Traditionally software was developed by implementing the methods and then testing each method. In comparison Test Driven Development (TDD) attempts to implement and specify the tests before implementation. This is achieved by iterating through a repetitive cycle of software development. This process makes it necessary to write a failing test case before implementing any methods. As a next step we then implement just enough code to make the test case pass and then refactor before repeating the cycle. The benefits to using this type of development in that the code written when using this method of development in often clearer and cleaner due to the fact we are only writing code that it necessary. The industry standard for the implementation of unit testing is Junit4. TDD is a fundamental concept of the agile methodology

For part 3 of the practical I used the EclEmma tool. This determines the code coverage of the test cases. When I ran the code coverage the first time I got 85% which means I didn't completely follow the TDD process although most of my code was covered the invalid triangle was the branches that was causing me issues. I realised after that my logic was slightly wrong as I had implemented the code, so it only failed when all the sides were 0 instead of if two sides are added and are less than the longest side as in this instance the sides would not connect. Once I fixed this the test passed. I tried to delete the 1 on the test cases and it resulted in the coverage dropping dramatically. So, I can say that there are not any redundant test cases within my code. The problems that can occur when redundant code is added to the test cases is that bug can be introduced as when we apply test cases the code we are testing may be covered by other methods this causes regression. In this case it is important to note that code coverage could be at 100% but it may never fully prevent bugs due to regression. In terms of coverage we needed to implement statement coverage where all statements are covered at least once Also due to the method containing if Else statements we needed to ensure the branch coverage was also covered. This is where my test had passed but on analysis of all the branches had not been covered this type of coverage is normally highlighted yellow in Eclipse, so it is easier to detect.

As a technique of developing software, I can see how using TDD can benefit the design of a project. However, I do think that it's more useful to utilize TDD in larger designs rather than in this assignment as it is quite simplistic. Nevertheless, it has been beneficial to observe the TDD process.