COMP47480- Contemporary Software Design

Learning Journal

Orla Cullen

14708689

UCD School of Computer Science

University College Dublin

May 2018

Table of Contents

1 Lab 1: Extreme Programming

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.

1.2 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.

2 Lab 2: Modelling with UML

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.

2.2 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.

3 Lab 3: Test Driven Development

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. Another type of testing is mutation testing this type of testing allows us to check that the functionality of the method/class hasn't changed and that it does the same thing.

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.

4 Lab 4: Object Oriented Principles

4.1 Reflection on my Learning on Object Oriented Principles

The first task in the tutorial was to alter the OCP class so it adheres to the rules set out in the open closed principle. To implement the open closed principle, we need to first understand to satisfactorily adhere to this the class or module should be open to extension but closed to alteration. We can extend the functionality of a module without necessarily changing the internal code of the class itself. Instead, we can create a subclass that extends the original class in this way the original code remains the same, but we can add more functionality. This is achieved by using abstract classes or in interfaces by using these we can make or classes more cohesive and less coupled. In the example code I implemented an abstract class Shape. This means that I can now have many classes extend by shape for instance triangle square and circle etc. By doing this we can also make some of the variable private, so they cannot be changed outside of the class.

The second task in the tutorial was to alter the SRP class so it adheres to the rules set out in the single responsibility principle. The single responsibility principle means that each class or module should have one responsibility. 'If a class has more than one responsibility it is overcomplicated, and the complexity is increased. Usually, this is because the classes have dependency on each other. Also, as a result the programmer may be giving functionality to a class that the object would not have in the case of the hexapod it represents both the human and the dog however the human should not be able to bark, and the dog shouldn't be able to throw the stick. Therefore, we can see clearly that the Hexapod class doesn't met the requirements of a single responsibility. Therefore, the hexapod class was split into two classes the human and the dog class giving only the methods that belong in each class to the respective classes.

The first two tasks of the assignment were relatively easy to understand and implement. The third task which was to alter the Demeter class I found this much more difficult to implement. I believe, I have been able to implement it as it should so that, it follows the laws of Demeter. These rules promote classes, that are loosely coupled because limits how much an object can know about its environment. In the original, Demeter code the shop keeper was able to directly access the customers wallet. However, the shopkeeper should ask for a payment and then the customer should check to see if they have enough and make the payment. Demeter in basic terms, means that it prevents the programmer accessing a third objects/classes method.

When we apply this to our example Demeter class the shopkeeper should be able to interact with the customer and the customer should be able to interact with the wallet. The shop keeper should never be able to interact with the wallet. This would make your classes easier to reuse ad you code will both look cleaner and be easier to test. Due to the way Demeter enforces its laws normally the classes written would have fewer errors and because there not really intertwined with other classes alterations in other classes are less likely to affect it. There are however some disadvantages to Demeter is that, if you need to make the third object do something then the clearest way might be to pass the third object into the method. It could also be implemented by providing another class that offers something like an actor interface that passes the request to the class required. This can make you code base larger and slower but on the other hand it will be infinitely easier to maintain and more portable.

5 Lab 5: Refactoring

5.1 Reflection on my Learning on Refactoring

Refactoring is an important part of the software design process which can assist in the improvement of your software design. This makes code easier and clearer to read and will assist in making the system easy to extend over time. Software that is poorly designed can often be confusing and lead to systems that take excessive amounts of time to extend or understand what the code is doing if meaningful names aren't given to methods in advance. This also allows other programmers to adjust the code without much previous knowledge of the code base.

This practical I found difficult just to follow the code through the refactoring process. It shows the importance of using meaningful names as the method and variable names and drives home that refactoring should be done as you go, for instance when you see something that doesn't tell you what it does or if an bug is introduced to the system or if you have a code smell then it would be beneficial to refactor.

This system suffers from giving a class and method too many responsibilities so the first thing I tried to do was to make classes of Car, Motorbike and All Terrain and have them extend the Vehicle class. The statement method in the Customer class was also doing too many things. This was broken down into a method that updates the frequent points and a method that gets the rental cost for the vehicle. This is a better as it allows us to erase the switch statement from the method. The ability to refactor the code allows us to come up with a solution to a problem and refactor it as we go creating a flexible and improved solution.

In this step we would also observe a code smells. These relate to parts of the code that aren't just in need of simple refactoring but redesigning to get rid of duplicate code and eliminate the issues related to the code. Common code smells like duplicated code we can extract the code and make it a method which can be called this allows us to use the code in many places with the implementation only be written once this is good practice as if we need to fix something in the duplicate code we now only have to change it in the method instead of finding every instance of the code. Feature envy is also common this is when a class is given a function that shouldn't be in the class. This leads to classes being coupled and being dependant on one another. Like duplicate code long methods can also be extracted to a separate method which gives a better understanding of the method.

Divergent change can occur where a class violates the single responsibility principle which means when we change one class it influences another class and again this can occur as what is known as shotgun surgery when you change something in one class that requires small changes in many classes. When dealing with currency or special strings these could be modelled as a class this allows us to control our use of primitives which could prove detrimental to the design if overly utilized.

Refactoring is an essential part of the software development process where it is not used regularly systems often contain code smells and naming issues that can make your system difficult to alter and expand. It is imperative to refactor code regularly, so your system can be kept simple, flexible and easy to understand.

I didn't follow the steps as I should have so I didn't get as far I could have with the refactoring had I explicitly followed the steps. I got rid of the switch statement and extended the vehicle class first, so it was a bit unclear then what to do next, so I don't think I have refactored the practical as I should.

6 Lab 6: The Observer Pattern

6.1 Reflection on my Learning on the Observer Pattern

This practical was on design patterns with respect to the observer pattern. This observer pattern tries to address the issue where classes can be too dependent on each other. These systems are what are known as tightly coupled systems. Consequently, by implementing the observer pattern it allows us to have a one to many relationships between classes by keeping a track of an observers list where the observers are notified if the state changes. In a nutshell design patterns just provide a way to solve a software design issue that occurs regularly. These patterns have been established time and again, so we know that they will work when we implement them and don't have to waste time trying to discover our own. The observer pattern when the Person is notified there is a change of state it broadcasts that change to all the observers in the list. It can be known as multicasting once the observers have received the state change they know about the updated state. This can be difficult to handle in large systems as the change cascades though the observers list and their dependant's bugs are easy to introduce so care must be taken. This is driven by events and has a somewhat publisher subscriber feel to it in the way the changes manifest in the system. There are two basic models of the observer pattern. The first is the push model I implemented this as a first step in the practical. The idea is that the person pushes the changed information to the observers list. However, sending all the changed information may not be a good solution as I suppose some of the observers may not be affected by the change. Afterward I then change the code to implement the pull model, personally I found the pull model more intuitive and easier to implement even though the push model is theoretically the simplest method. In comparison the pull method tells the observers list a change has occurred. It is then up to the observers to then ask what has changed within the system and update the changes as needed.

There are a few matters to consider when implementing this pattern first we need a way to have a map of the interactions between subject -> observers we can do this by creating a list of observers in the Alarm Clock class. This holds a reference to all the observer objects that will need to be updated. We need to also ensure that the observers are correctly detached from the subject to ensure that the observer can be collected by the garbage collection when not needed anymore. It is possible to observer multiple subjects in this case the observer needs to know where the update is coming from, a way that we can achieve this is to pass a refence to itself with the update message. Message delivery is not ensured by this pattern. The observer pattern plays a key role in the model view controller pattern which is utilized in many GUI based designs. Subsequently, it is a key architectural pattern utilized in many systems and libraries.

7 Seminar Series: How we build software

7.1 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.

7.2 Seminar 2: Facebook

This seminar was given by Richard who is a network production engineer and Mike who works in Data centre management. This seminar focused on the operations side of Facebook more so than a on the software development aspects. Facebook has roughly 2.1 billion users not counting the subsidiaries, WhatsApp with 1.5 billion, Messenger with 1.3 billion and Instagram at 800 million. In general, the data centres rely heavily on cooling and part of Mikes' job is to manage and maintain the servers. Failure is inevitable in software and operations however the most important thing is how you respond to the failure and n what you can learn from the failure these are key aspects to the operation of Facebook. If an individual finds an error in Facebook they are encouraged to flag it to the correct team and hand off the issue. Collaboration and communication and learning from mistakes appear to be a very high priority in the Facebook infrastructure.

Software Methodology:

The software methodology that is applied in Facebook is based from an agile framework and the tools and methods that are used are very flexible. Frequently, teams choose how they wish to develop the software and what tools they need. Facebook don't appear to be too concerned about how the coding is done and what methodology is utilised if productiveness and the code written benefits the system. In this methodology they will also ensure that they have some automated systems that will quantify and analyse the software developed for errors. In the event of errors, they have four teams the first will work together to keep the site running the second trees to reproduce the failure at a smaller scale the third will access the code for configuration changes and the fourth will dig through the data.

Use of Modelling:

Mike and Richard deal with the operations side of Facebook more than software development and as such each team decides on whether they use modelling. However, in their experience Facebook operates by form of effective communication not formal documentation. Their development process is much more cantered on Hackathons, social gatherings, and interaction within teams.

Testing:

It is important to note that testing won't always catch all the issues. They utilize unit test case which make up the bulk of tests by testing each component separately. This only tells us the component is error free it doesn't tell us if it integrates well into the current system. Integration tests are also utilized although there is less of these than the unit test cases. Integration tests are more complex as they test how well the components operate together. Another type of test that is implemented is Load testing. This is an important aspect in the testing process as it allows you to test the load on the system by simulating a peak environment. This gives an idea if concurrency is affected and the quality of the service being provided during high usage. They roll out changes in the system multiple times a day this ensures that the system is less likely to collapse and if it does the error can be found much quicker because less code must be analysed to find what caused the system to fail. In terms of testing to reduce the number of tests needing to be completed before rolling out they only test the methods that will be affected and not the whole system.

Software Quality & Refactoring

Facebook use to have a monitoring team however in recent years they have delegated the tasks of monitoring components to the teams involved. The teams are responsible for having alerts in place to make the developers aware of problems with the code. Facebook tries to automate a lot of these process as humans are generally unreliable. In this sense teams would have an incident manager which helps to steer the team the right direction to address code quality. They believe in getting thing done so code quality may not always be a high priority especially if they the time frame for deployment is quite short in comparison with the time thought needed to do the coding to perfection. This is where refactoring would happen, and this would be made a priority where features and updates would cease for a cycle to refactor the code. It is important to maintain the code as it allows others to join a team easier than if the codebase is ugly and hard to understand.

7.3 Seminar 3: FoodCloud

This seminar was given by Roy Philips who is the chief technology officer at FoodCloud. FoodCloud was set up to solve the problem of food wastage that occurs daily in supermarket chains. It is a system that acts as an intermediary between the supermarket and a charity to reduce the cost of food disposal by creating an incentive for the supermarket to donate food near the expiry to charities. Food cloud is a way of managing the donations process. The incentive that draws supermarkets into donating food is that it both creates positive media profile because they are donating to a good cause, it has a positive effect on the environment because food doesn't need to go to a waste pile and it is also alleviates cost of food disposal as it's cheaper for companies to have someone eat surplus food. One of the key aspects of this platform is its fairness algorithm which is hoped tries to ensure the same charities does not always get the best foods or the same foods. This means every charity has equal opportunity to benefit from the donations. Another very clever aspect was to include an impact interface which allows companies to visualize how much food was donated by individual stores and quantify the amount of meals it relates to when the charities utilize their donations

Software Methodology:

The software methodology that is applied in FoodCloud is based from an agile framework and the tools and methods that are used are very flexible. Scrum Pair Programming and Kanban which is a process management tool are all utilized. Kanban allows programmers to pull work as the programmer's capacity permit rather than when it is requested. Food cloud operate using a small 7-person team each is responsible for their own area, so the android developer has control over the android app and so on. They also work on stories which are like tasks where they only take on at most 4 stories at a time. This platform is in its infancy, so their architecture is very much being developed on the fly with adjustments being made at each iteration. Originally it was built as a dual, module with one module acting as the donations processing centre and the other taking care of the communications gateway between the donator and the charity. However, due to more traffic on the servers and more requests these 2 modules would be later abstracted out to create more modules that adhere better to the single responsibility principles. This will allow the future platform to scale more effectively as it gets more users form home and abroad. This platform is written in Scala and ReactJS makes use of the Heroku platform as a service which helps in the management of databases and it also makes use of the Akka streams which adopts the let it crash at has set instructions for recovering from a crash.

Use of Modelling:

Roy uses google slides as a mock up tool to make a more informal UML diagram to give a fundamental basic concept of what the team are trying to achieve overall. Google slides is a nice way of Interacting with other members of the team which can be useful when sharing ideas or making comments and suggestions on the overall design of the FoodCloud platform. These informal diagrams provide a basic idea of how the system should work to the other developers, so they can provide the iOS and android apps in a similar fashion. Their development process tries not to predict the future or make estimates of unknown factors because a lot of what they are achieving is can't be summarized in hard and fast knowledge. It is much more continuous and changing environment.

Testing:

Test coverage was originally very good but as the relational database grew it became very slow to test and subsequently this was ignored for some time. However more recently they have switched to H2 database which supports the original PostgreSQL framework this makes it easier to test because in memory table can be created as well as disk-based tables. Now they have good test coverage again which is kept up to date. They have been quite lucky over their development that they haven't encountered too many bugs. So, it has worked well for them.

Software Quality & Refactoring

Software quality and refactoring is less of an issue in a platform like this due to the size of the team and the fact that each person is fundamentally responsible for an aspect of the platform they are usually writing code for each part on their own or maybe some pair programming. The refactoring process is done but especially now where they are abstracting out the modules to adhere to the single responsibility principle. However, they really work on a make it work make it elegant then forget about it. Where possible the rule is they try and make the methods as meaningful as possible as they go and correct/refactor as they go which is just part of the continuous develop and deploy cycle.

General Comments

Roy enjoys working in a small team as its quite focused on what you are responsible for and there is a lot of creative control when there are less parties involved. FoodCloud development process is way more unstructured than that of large companies this is probably due to mainly one to two people working on a module. This means that these developers have massive knowledge on the internal working of the system and are highly skilled in their area. They collaborate but from a perspective of the general direction and flow of the project and can assist each other in coding aspects where needed but at the end of the day for most of the time they are delivering massive chunks of the platform by using small amounts of pair programming but mostly solo programming.

7.4 Comparison of the Seminars

After attending the how we build it seminar series I will now look at the similarities and the differences between the companies as presented by the representative providing the seminar. Subsequently, I can say that from my observation I believe that seminar 1 IBM and seminar 3 FoodCloud portrayed the most similarity. This is probably due in part to the fact that they talked mainly from a point of view of how the develop software from their software methodology to their use of tools. In comparison Facebook talked mostly about the operation s side of the company and its data centre management than on the process of the software development within the company.

IBM are probably the most structured of all 3 seminars and provided a very thorough view of how the develop software and the quality control process that the software goes through after each sprint. They have a set framework of how they operate and like everyone to be coding using the same development environments. They also require developers to write test suite for their code and utilize tools such as UML. The code is review by the team lead or someone senior this ensures that issues are caught quickly. Although at IBM you work in small team, they would need to be very structured to ensure that code would integrate well compared to Food cloud who essentially do some pair programming but a lot of the time the individual is responsible for the whole outcome of an aspect of the project. In some ways Food Cloud would allow more creativity since you really are programming alone or at times in pairs, so the code only needs to be readable to you the programmer. At IBM they are a much bigger company more people will be working on the same code base therefore the code needs to be of a certain quality to allow other programmers to easily work on the code base therefore they are much more systematic in their process.

At food cloud they are the masters of their code and the likelihood of others reading or working on it is far slimmer. At FoodCloud they essentially try to make the system work at times the system looked quite hacked together it seemed that the code was radically refactored to make it a more elegant design once they had they concept cemented. Where at IBM it was systematic and well thought through with lots of UML diagrams and Scrum meetings to communicate the system design to all parties clearly.

Facebook and Food Cloud also hold some similarity with both not really minding how the developers go about creating their solution if it benefits them, and you are productive they don't necessarily specify the development environment you must use like IBM. Facebook also seems a bit more social in that they would often meet over coffee or drinks and effectively communicate ideas than rely on formal documentation. The key thing I will take away from these seminars is collaboration is paramount to be able to work from an agile type framework which most of these companies employ. Similarly, communication is vital in put across your ideas, you, and it is inevitable that you will fail or make a mistake in your career. Thus, being resilient and able to assimilate and learn from mistakes is a is essential in developing a career in software development.

In comparison to some of the course materials IBM showed most similarity to this course touching on all the tools that the use from code coverage and analysis down to the systematic way they go about refactoring. Where FoodCloud codes in a more like how I have programmed over the course of the four years much more get it done and then fix it up. FoodCloud even referred to the single responsibility principle as being a reason behind abstracting out the system into more modules, so they are more loosely coupled. This seminar series along with the course materials have given me a better understanding of software development in industry and overall, I feel the knowledge gained throughout the course will benefit my career.