COMP 47480 Contemporary Software Development

Lab Journal

Gregoire Cousin (18204188)

Masters in Computer Science

UCD School of Computer Science

University College Dublin

04/26/19

Table of Contents

INTRODUCTION

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE
GREGOIRE COUSIN STUDENT NUMBER: 18204188

Work done: all

Throughout our time in class, we dove deep into multiple approaches and concepts of building software with proper structure. This was to understand the fundamentals of Agile, Scrum, Traditional software methodologies like code and fix and the Waterfall Process to name a few. We looked at the best methods to properly detail the specifications and implementation of a project.

I found the Waterfall Process Interesting since it was a widely used method in history. It meant going through the entire application : from analysis, design, coding, testing, and maintenance. This was done when working in teams and the process could take multiple weeks and more.

When working in a team, it is essential to check for errors and manage them throughout your project. This is why we have to work backward from the Waterfall Process. As programmers, we found better technics including Iterative and Incremental methodologies. These methodologies consisted of working in iterations throughout your project or finishing the whole application in one go before maintenance, testing or deployment. Each process has their individual pro and con depending on the type of project at hand.

If you needed to build features and have direct access to your customers rather quickly, the Incremental method is a better approach. However, if you didn't have all of the details in front of you and needed to code the program/idea over time, the Iterative method will suit the project better.

In class, we also learned a lot about Agile and how to incorporate it within a teams project. We then compared it with the waterfall process. We found that the Agile method had better outcomes in interactions between the team themselves and their customers, building software that worked well, and a more flexible way to deal with potential unexpected changes and errors.

In Agile, we learned about extreme programming, user-stories, different meeting methods such as the planning game and stand-ups. The Agile process also helps the ability to structure a good testing environment. Software testing is vital to the process of building a useful application, and many testing technics are made to work hand in hand with Agile methods. We learned different testing methods such as Unit-Cases, Refactoring, Code Coverage, Mock Objects and what these terminologies represent.

We completed an assignment based on a video that showed the interaction of senior developers and the person responsible for the project. (This would typically be the project manager.) The project had an issue and new errors needed to be fixed. However, the testing environment was wiped from the project since the team believed they had finished all test cases successfully.

Everyone in the team realized this was not the case. The exercise was excellent to go through since it consisted of learning what could potentially go wrong when not following proper protocol. Losing test files while working on a project might be viewed as very wrong and a bad practice to fallow before being entirely sure that the software is out of your hands. Even then, it is good to keep track of your work with a history management tool such as git for any potential further work, future releases or important deployment. The video also showed us to be extremely meticulous and detail oriented with the way we approach building software, taking account all of Agile's processes, project planning, the importance of each phase and keep in mind what their values are.

I believe that structure and communication between your client and your team to be essential in the process of building good software. This means that a proper approach tailored to your program and team is necessary. In our modern ways of working in development, Agile stand out to be the best and most advantageous of team-based software building processes.

UML AND MODELING

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE
GREGOIRE COUSIN STUDENT NUMBER: 18204188
COMP 47480 PRACTICAL

Work done: A Summary

In the work that my teammate and I have done, we have found that a lot of principles of UML consists of ways you organize your code. This theory implies that you are merely getting closer and closer to your goals as a developer or team. By sectioning off behaviors into boxes and relating them with other ones you can completely map out visually what you intend to accomplish in the end. UML can also be perceived as a method of communication, between you and a small team but also facilitate a large one. In our work, we have found that the more technical you get with the diagrams, the better you are able to gather the details needed at hand to complete the jobs. We were given a set of task to relate a machine with and a shopper and simulate what it would look like if you were to create a Case Modeling diagram of what that would look like. We then were instructed to find the positives and negatives of an example based class modeling diagram. Since UML is like a map to guide you from point a to the last step, it is essential to understand what your constraints are. We then dove into the different methods and approaches of sequence diagrams. This type of layout relates and concentrates on the timing of the project at hand based on behavior and functionality.

Reflections: Why, alternative, preferred, Linkages, areas of computer science, deeper study of some aspects

# Case Modeling

In Case Modeling, the structure is based on simple key points of interaction between two or more sources. It would be easy to take something that we interact with and thoroughly analyze its components, behavior, and functionality into everything it's capable of doing. However, this model gives us the ability to gather data more in an eagle eye point of view. Take the sequence of events that you would need to go through when interacting with an ATM. You could describe all of the things that go wrong along with representing all of the steps. However, with Case Modeling, you need to have the eye of someone external. Instead of explaining what would happen when you type a pin wrong, all you need to describe is that you typed in a pin. Case Modeling in its purest form, are the key of events that occur when dealing with interactions. As if you were someone from an external source looking in.

# Class Modeling

I find going through the Class Diagrams approach to be the most simplistic way to deal with an abstract problem. To me, Class Diagrams are clear and concise. It has the simplicity of working with small chunks as if each class was viewed like a posit note but also the complexity of joining them together to explain the relationship of what's inside a specific class. Another cool aspect of dealing with class diagrams is the fact that we can visually see what goes on inside of an application but also translate it to code through the use of specific software that is based on translating the diagram into code. Class diagrams have useful arrows and other visual tools making it easy for developers to associate classes together. This can facilitate the understanding of a multitude of complex concepts in the software such as inheritance, general relationships, aggregation and so on. Since we have found that object-oriented programming to be beneficial to organizing and building software, Class Modeling attempts to give you a visual way to work with objects, their attributes, and internal functions.

# Sequence Diagrams

In sequence Diagrams, we are less likely to focus on the internal side of who something works the way it does, or how behavior affects one particular item over another in the sequence of events that it would. Sequence Diagram focuses directly on the WHAT will happen throughout the behavior of an interaction. This means taking a whole organism and separating it into what it does over time. Take for instance starting a car. When we start an engine, we first have to turn the key to tell the vehicle to activate the motor. In this case, when drawing out what a decency diagram does; we would first need to turn the key and know that the motor will react to start the engine but then relate everything back to the key since it’s the interacting tool that got us there in the first place.

# Conclusions

In my opinion, I believe that all three modeling practices are necessary and essential to use as a unit. It would depend on the application at hand or even what part of the application process would need what diagram the most. We have seen OOP be very useful. The class diagram method would resemble OOP perfectly but give you less focus on the sequence of the application. To accomplish this better, adding a sequence diagram would benefit your application. Since these are complex methods, we also need a way to communicate our ideas effectively. Which is where Case Modeling would strive.

GER AND MONGO DB

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

Introduction:

Ger Hartnett is officially the VP of Engineering for MongoDBs headquarters where he manages multiple teams across Europe. He has been part of the company for the last 5 years dealing with a variety of tasks based on the mix of technical and management jobs. His talk today was based on his upbringing to his current position, his struggles and his accomplishments.

Education:

He started his education in Limerick then attended UCC from 1992 to 1995. Though his education Ger learned programming principles and worked hard on improving his technical skills. He developed an understanding for technologies in CON, UNIX and EMACS. Ger was very interested in the building of programs but also the fundamental reasons to why he coded. This meant learning a lot of theory.

Career:

He began his career by interning at Analog Devices in 1985. After a year of learning how a company works and gathering new principles and personal beliefs, he went on to work at SW Engineer as a QC data engineer. At this stage, Ger wanted to be part of start-up companies where he landed his first job working for a boss he believed to be crazy. He then tried to work for another startup, but the company failed. From these experiences, Ger knew it became essential to learn when to leave a company and how to get back on your feet. He quickly moved up to be employed at Motorola where he was given the opportunity to work on some of the best projects of his career. This included developing GSM chips that are implemented in our phones today along with working side by side with IBM.

Ger gathered project management skills because of his relations with IBM. In this period Ger also learned a lot about what it meant to debug your code and reuse the same tools that were given at hand rather than restart a project from scratch when it has issues. Since agile wasn't highly recognized at the time, the company was having a hard time keeping up with the quality of their project. Everything based on management was dysfunctional for the company. They tried going from a traditional approach such as models based on UML all the was to an Ad Hoc method. However, a team of 7 people including Ger gathered their skills together and found that the principles based on agile to be a good middle ground and saved the company. This is where he understood that it doesn't matter madder how many people you have in your team. If you have 7 and all members are focused, then you can accomplish even more than if you had 70 employees and a disorganized project.

Ger then moved on to other companies such as Digital Equipment Corporation and Tellabs where he worked as a Software Engineer. He hit a big break when he landed a job working directly with Intel where he worked with technologies such as Microcode, Linux Kernel, and other device drivers. Throughout his career, at Intel, he worked closely with the Waterfall model along with other agile methods. After eight years at Intel he started his own company: Goshido. Goshido was a company that helped customers organize their current projects. Throughout this experience, not only was he able to work directly with shareholders, investors, and customers but also learn from which of these experiences he got the best results from when directing a company. Ger believed that concentrating on your customers needs to be a priority. Seven years later Ger left his company and went on to work for MongoDB.

Other Moto:

In his Career Ger gathered a lot of understanding on how to become a good employer and a few rules to live by when working for a company. These rules included staying away from unfair bosses but remain composed at your job, knowing when to stop a company when the budget is too low and having a backup plan, communicate often between your peers, that 90% of startups fail and being ready for action if they do.

SOFTWARE TESTING

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

Introduction:

In our lab today we dove into a few concepts of software testing that helped me realize how valuable it can be. In software testing, many different methods and approaches chosen are based on the code at hand. In our lab, we used standard tools such as java Junit but could have as well used TestNG which are both platforms that provide annotations to identify test methods, assertions for testing expected results, automatic testing to give you accuracy in your code and generate quality reports of your test results based on UNIT TESTING. Unit testing is where you test specific parts of a system to verify that the behavior is what you want it to do. The JUnit API is loaded with multiple assert type functions such as checking for an equal object, variable, boolean values to make sure you are passing the right information.

A Bit of Theory and Methods:

There are many different types of software coverage methods, and each one can work affectively when testing alongside another. To start, we can provide our code with a lot of results when testing the functionality of a program through branch coverage. Branch coverage is all about the decisions making of an application. Making sure that each possible branch is checked and ensuring that reachable code is executed. We can also test code with Path Coverage where unlike branch coverage, Path Coverage will cover every possible control flow paths through a program. These are some of the most popular methods of testing code.

The Program:

In the assignment, we were asked to provide knowledge on how to test if the given output of a class was precisely the same String in the test class. In the main class, I wanted to test most of my outputs based on a large if statement. Checking if a given number was divisible by 3 or 5 independently or if they were both divisible by 3 and 5. We were asked to simulate a test-driven development. This meant that the tests were needed to be given to the professor one by one; reanalyzing the code at each interval, making sure they pass and committing them on a git branch. It was a good exercise that sets you up for the real world when testing. We also were given the task of creating some a sort of mutation based testing from the previous code written. Mutation testing is checking the polar opposite of what you intend to write.

Emma and a lot more:

Since it is a good practice to use different technics of testing, we used Emma to give us an overview of the coverage achieved by our test cases. This was a graphical view with percentages and line highlights of where the test reached your code in specific lines and where it did not. I found this to work well with Maven, so I took the liberty to learn to implement Emma, jUnit and generate tests based on Mavens architecture all in one module.

8BYTES

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

Introduction:

Today we had the chance to talk to Peter, the CO-Owner of 8Bytes about his business 8 Bytes is a company that is based in Dublin but deals with significant enhancements in the world of software engineering. The primary role of the company is to provide guided help to people who want to start a start-up. 8Bytes stands by many different important methods that help the team run efficiently. They have helped over 1000 companies from the ground up which also consisted of leading some unsavory people. It all started when a group of friends from college got together. Eoin, Brian, Adam, and Peter are the members of 8 bytes. During the talk, Peter talked about how it was essential to keep a close group of friends and if you had to build a team to make sure it's 1small enough as a start.

History:

Peters has a long line of opportunities that have been given to him. While he was in college, he had multiple offers from Hub-spot and other companies to leave college to work for them as an intern. However, this was not in Peters vision of interest. He decided to finish college and work on a couple of projects based in Ireland and America where he was offered a lot of offers from clients of his first company. This company was based on a Fitness App. Throughout this process, he worked with other companies such as TG4. He developed his skills in android, server technologies, Kotlin among others. Before starting his own company with his college friends, Peter interned for intercom for a while. Where he was exposed to testing principles and how working with a small team is a good idea.

Principles:

8Bytes works with a couple of good principles.

The list that they stand by is:
 - *communication, development process, testing, maintenance and specification.*

Peter has said that when dealing with specification of a company, it's a good idea to keep a good log of everything that is needed in the end and to break down parts to its smallest details. While doing this, making sure to keep a good understanding and communicate effectively with your client is a good idea. Also, when dealing with specifications be aware of pre-conceptions, make sure you have a contract that highlights everything and figure out the deadlines. While the development process if crucial to building an application for a client, testing and monitoring the application can make the building process smoother. Peter states that maintaining an application can be quite tricky and time consuming. Every process needs to be detailed.

WATSON CARE

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

Introduction:

Today, we were thankful to be afforded the opportunity to speak to Marie, the IBM Lead in Watson Healthcare. Graduating with an advanced degree in Software Engineering at UCD, she has thoroughly enjoyed working within her field for the past fourteen years. Using the concept of innovation in order to solve some of the hardest issues that we are facing through data and cognitive science, Watson Care strives to improve the lives of many. They offer an impressive SaaS platform which is business orientated in nature; therefore, their application is readily used as a means of managing care across clinical, human service and judicial organizations. In recent years, Watson Care has not only assisted myriad organizations within America, but has also both helped the homeless populace across the world and aided the survivors of forest fires. Watson Care is more than just a software platform; it's an organization that strives to provide an easy way to ensure that their product is accessible, perfect for the consumer and works to deliver ongoing support to their clients after the product has been sold. In this way, Watson Care is evolving their product based on the needs of today's clientele.

History:

IBM has built a very solid reputation throughout the years, especially in relation to their ability to adapt to the needs of myriad clients and their specific situations. They aim to help many innovations with a set of life cycles including the act of building a vision, planning the ways in which they should enact this vision, beginning the development, delivering on this and both operating and supporting their client's products after the fact. Marie states that their modern model is not what IBM has always employed throughout their history. She remembers a time when teams would make use of the waterfall model, and thus, they would feel extremely stressed during the entirety of the process; stress would only become relief once everything was finalized. When Marie was first hired for her current role, she flew down to Texas in southern America where she trained in order to learn this modern model; here, she learned about tools such as Muraly and Trello. These are both tools and applications that follow, and then incorporate, the current agile processes that are so apparent in modern times.

IBM Model:

At IBM Watson Care, the team uses a set of tools that work to assist them throughout their stages of planning. These tools include:

1. RTC- This is an application that successfully generates UML diagrams for their various projects.

2. Epics- This is a way to both give an application a theme, and accurately view the overall picture of any given application through the use of milestones.

3. User Stories- This helps to utilize post-it notes.

4. They are extremely thorough throughout the stages of documentation, testing and development.

Moreover, the planning stage also requires the company to design diagrams in order to assist employees in understanding the general workflow of their project. These diagrams include a planning matrix, which works to both detail the level of accuracy within the software, and which aspects of the software require additional work. These diagrams include both high and low-level functioning components, and various other flow-based diagrams which can both be generic and customized in order to fit a specific need.

Watson Care utilizes these methods of documentation rigorously; for instance, they might employ a web-based format in order to ensure that it is accessible for the new customers of any given company. On the other hand, they might provide over 120 pages of documentation for every application as a means of making sure that their customers receive extremely thorough and high-quality work. Considering that IBM aims to be precise above all else, they have a constant need to adapt their old documents in relation to any new implementation.

Marie believes that a two-week sprint is often longer than necessary for the majority of IBM's jobs, but also considers it to be enough time to both focus on the fine details, and sell high-quality products to its customers.

Technology:

#workflow

Watson Care makes use of technology such as Docker for its deployment, Postman as a restful tool, apache for any web-based tasks, eclipse as an IDE and Chef for its deployment scripting. As the company has a significant amount of data both flowing in and out, they need to make use of a very robust database system; this is where DB2 and H2 come into play.

#development and testing
Watson Care employs a variety of languages, based on their many services: these include Java, JS, Graddle, Groovy, Ruby and Python along with shell-scripting languages such as rvms. Since each and every part of the code requires testing to ensure accuracy, IMB has a vast set of technologies that they like to utilize. For instance, their test-based tools include Junit in conjunction with java, Mohito, PMD, find bugs, sonar cube and code review. Furthermore, IBM also uses Check Style in order to control the varying complexity of their code through a cyclomatic method; usually, this method would ensure that they can successfully check whether loops that are implemented within the code are too high and, therefore, need to be adjusted.

#deployment
Adjusting a large set of code files, and simultaneously organizing your team to ensure that this is successfully done, is certainly not as easy as it sounds! A faction of Watson Care is specifically designated to manage the upgrading, policing and integration of builds along with deploying these to a VM. Moreover, the aforementioned team is also responsible for both testing features, and guiding the outer-automation principles that are needed for the application to be the best it can possibly be. They aim to gather feedback from their customer and user-base before the program is officially released, thus ensuring that all of the features are specifically tailored to these individuals.

Security:

During the process of each sprint within their process of development, IBM aims to provide both quality and integrity to both the core principle within their application, and the people at large. Considering that the company is highly regulated in order to ensure that they stay within the bounds of contemporary society and its norms, IBM restricts the visibility of their content. This applies to specific teams within the company as well as the general public, as Watson Care is a platform in which users often provide highly sensitive data including doctor's notes, treatment plans and personal conversations; thus, the company is trusted with data that requires protection at a professional level. ComOps are a security-based faction within IBM that works to manage the production of the company's software: they use tools such as Elastic Search, Kibana, Logstack and VHI. What's more, they fulfill a plethora of differing roles such as updating customers on a live platform, and because of this, they are unable to see the customer's sensitive information.

Support:

Watson Care provides high-quality support to their customers through the use of three separate levels of technological support. The first level provides customers with service requirements, while the second provides specific solutions in managing their records. In addition, the third level involves thorough technical support from a talented developer.

Conclusion:

IBM is an extremely large company, complete with core moral principles that have given them the ability to be up amongst the top influencers of our generation! With a whopping 26 years of boasting the very best patents, translation and legal clearances across the board complete with a brilliant and accurate security review, they certainly deserve their acclaim. IBMs company, Watson Care, is constantly changing, scaling and evaluating current technology on a global basis. They stand by the mantra that a bigger company does not always make for a better team, but in fact, it is the core ideal of trust between their workers, compliance and consistencies that ensure that the company is strong enough to sustain their glowing reputation for many years to come.

REFACTORING

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

INTRODUCTION:

In this lab we were given the instruction to explore the possibilities of refactoring code and understand all that deals with it. Code smells are the root of defining what a problem can exist in a program at given sections. To identify a code smell within a piece of code you need to first understand the framework in which you are working on and what the regulations are.

CODE SMELLS:

Code smells can come from a verity of categories such as Bloaters which are code smells based on the accumulation of unnecessary code. These code smell names include Long Methods, Large classes, primitive Obsession, Data dumps and so on. Objective Orientation Abusers are another type of code smell that deal with the incomplete and incorrect state of a program. Objective Orientation Abusers include switch statement faults, temporary fields, refused bequests and class and interface differences. Other categories of Code smells are Change preventers, Disposables and couplers. Change preventers deal with refactoring one place of code that eventually will lead you to changing the same principle in other sections of your code. A good example of this would be the Shotgun Surgery where you need to change a whole lot of variables at once. Where as Dispensable are based on pointless code such as over commenting, lazy classes and duplicated code. The Last Category is based on Coupling. This code smell includes extensive coupling between classes. A good example of this category is the feature envy. feature envy is where a smell occurs after fields are moved to a data class. Moving the operation to this class will be necessary in the refactoring period.

REFACTORING:

Refactoring code plays a big role in the creating or a program and should be respected by developers. The means that if another developer has refactored peaces of code but has handed off the code to another, it should not be touched or be limited to change without communication. This is said with the understanding that refactoring happens on a case by case basis to give code clarity. This might mean that a developer would be obliged to extract a local, constant or other field based variables, change and extract a long methods and class function, use the pull up or push down method or even extract interfaces to organize the schematic of the program. Code clarity is vital when it comes to building a good program since the goal is to provide concise and well organized programs. This increases the ability to reuse segments and expand on particular sections of your code with a more explicit manner. Code issues can also happen in unorganized code fragments. This means that is you have lines of code not embedded in a class, for in a method oof some sort you can damage the ability to develop future features to your program. This is very important to avoid when working with OOP. When extracting a method. it is normally because a method can be too long or has unnecessary fragments. this can decrease readability for a program and make functionalities and features confusing. Since we read programs like documents it is important to take in consideration what we are extracting and refactoring. This means that renaming variables is highly recommended to provide sense to the programs functionality. Every IDE has the potential to read segments oof code and analyze repetition by a find function within the IDE and give you the ability to change code via a tool set. Code duplication is one of the most common types of code smells. This can be seen by redeclaring the same variable which can be easy to spot and refactor but in a more difficult setting code duplication can also exist in the same functionality of a classes behavior. Another popular reason developers find themself refactoring is through finding conditional code statements confusing. This means that a while or for loop statement would cost more load on the desired functionality than is the developer had used a switch statement. Therefore, the developer needs to refactor the appropriate segments and match it with the right function.

DESIGN PATTERNS

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

INTRODUCTION TO DESIGN PATTERNS:

Before we dive into the characteristics of observer or strategy patterns we need to understand a lot more about the importance of design patterns, what they are and what they are used for. Design patterns is a template for how to solve a particular problem. A design pattern can be used in many different situations and there are usually set rules for each one of them. It uses common code practices such as Inheritance, Encapsulation, Instances a local and external variables to accomplish its particular goal. The most common design patterns include the abstract factory, singleton, strategy, observer and template methods.

OBSERVER:

As an introduction to one of the most common design patterns, we will mostly talk about the observer and strategy patterns. The observer pattern is defined as one to many dependencies between objects that said that every time an object changes its state all dependencies are notified and automatically updated. The observer pattern uses encapsulation to produce this task. It encapsulated the core elements based on a subject abstraction and its variable components in an observer based system. This is closely related to the MVC idea. This pattern is mostly used when an application need to scale more than it already is. Let's take for instance a bank, a mailing system in which a banker is responsible for and a customer. The bank holds the customers money however the customer still needs to have some sort of control knowing how much money he has in his account since he buys mostly online. The bank account is linked to another few family members who also need to know this as fast as possible. This is where we have a mailing system that checks when a transaction is being processed from the bank (in this case our interface that sits
between the subject and the customer) the mailing system gathers the data from the bank and updates all the family members. This would be the same things in design pattern. Where you would have to have a subject and interface which gathers the information form the client and informs the customers how much money they've taken out to be taken out and then also updates the other family members.

STRATEGY:

The other commonly used pattern is the Strategy Pattern. This describes a family of algorithms, encapsulates each one and and makes them interchangeable. This makes it so the algorithm at use varies independently from its client. If we have an open closed principle issue with our program, this pattern would help fix a lot of issues since its main focus is isolation of different components. The strategy pattern helps figure out what behaviors are meant to be used from derived classes separates them. However, we sometimes want to come out with a goal that are shared between these different behaviors. Let's take for instance a dog and a bird. The dog does not have the ability to fly but is able to walk while the bird can do both. In this case it would be great to create a flies interface that are both connected to two other other classes a can't fly and

a can fly class. Since our client is the animal in this case we can dynamically change the animals behavior by connecting it to the interface as an instance variable. In this case we can now create many different types of animals without affecting any sub classes. It becomes very interesting when you are able to mix both the strategy patten and the observer. In out case if a bird decides that it needs to start flying the animal is notified creating a mechanism that is based on how everything works around us.

FINAL CONCLUSION : COMPARISON

EMAIL: GREGOIRE.COUSIN@UCDCONNECT.IE;
GREGOIRE COUSIN STUDENT NUMBER: 18204188; COMP 47480 PRACTICAL

Introduction and Companies:

Contemporary Software development is the understanding on the correct structure of how programs take their course, their evolution and the practice of modern models used in the industry. Throughout the course we dove into the difference in Ideology, abstract differences and approaches between three companies. The companies have their own differences when approaching software design. This can even have an effect based on if the company is a startup or not. The companies include 8BYTES; a startup created by a group of four that share common interest to help someones idea to become a reality with the use of providing software solutions. IMB; a global wide company that has the reputation of starting some of the foundation pattens which moulded the software industry. IBM provides services for countless businesses along with having a part in all aspects including: the banking industry, healthcare, security and other necessary processes in society. The last company we learned about was MongoDB. Like IBM, Mongodb attempts to provide a world wide solution for management, structure and models based on the gathering of data.

Ideology and Differences:

8BYTES focuses on the advantages of being a small company. This means that since they are reachable 8BYTES is able to consult its clients in a one to one basis. This direct contact makes the agile process simple and attractive for its partners. Since 8BYTES isn't a big company it doesn't need to rely on the most advanced schema and planning design tools but rather focus on executing which can stream line the process of building software. We're as a company such as IBM doesn't have all the resources to touch base to everyone making the planning process more necessary. It is also important to note that since IBM and Mongo are larger companies with more employers, they are able to work on bigger projects than 8BYTES would. That being said, 8BYTES still has a lot of similarities on its processes to how IBM would operate. They still rely on the Lifecycles of sprints and use the agile and scrum standards. Mongodb plays a big role in our development of software but since the company provides more of a tool than a multitude of software programs for individuals and businesses needs. The company strives on the expansion of its own software rather than the focus of accommodating different software and agile principles.

With a larger Team you can gather more knowledge and build a system where more tools are used. Since the tools help employers be creative, this is rather important. IBM strives to have designated teams that use different tools. However, with a larger company this introduces a barrier in communication between the teams working on the same project. 8BYTES can have a lot more work in learning new ways to tackle a problem. When they need to accomplish something, they can't send it to another team. This gives them more responsibility but since they are a small company, communication between the team members is easier accomplished.

Conclusion:

Between these companies, there has been a lot to learn when it comes to different approaches to run a company. A large company would have its advantages in being able to reach a whole variety of customers but would have its difficulties with executing in detail based on the customers needs. This would be due to the lack of direct contact throughout the building of the program where as a company like 8BYTES would strive on communication but would lack on a demography of projects outside of its proximity.