INDIVIDUAL REPORT

COMP-8117

Advanced Software Engineering Topics

SCHOOL OF COMPUTER SCIENCE

INSTRUCTOR: Dr. Usama Mir

Author: Abhishek Kesiraju

Student ID: 110068881

Feels like yesterday when I sat for my first class for Programming with C in my Undergraduate course. A 17-year-old individual inspired to take up Computer Science by watching Iron Man's Jarvis, his personalised Artificial Intelligence assistant. It is after having an introduction to data-structures, object-oriented programming concepts, web designing and Database management Systems that I have first heard of the word "Software Engineering". At this point, I only knew learning all these subjects would make me a Software Engineer in corporate. Nothing but a mere role of how an individual is referred to as, when he completes his Computer Science and Engineering course. Little did I know that the subject would become a guide to how the world of Software runs. As any other student who would be excited about his course, I skimmed through my official course textbook "Roger Pressman and Bruce Maxim, Software Engineering: A Practitioner's Approach, 8th edition" and found the highlighted definition of Software Engineering –

The Application of systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software. – IEEE[IEE93]

My first understanding when I read this was, okay maybe just implementing software or being able to code with good documentation in an organized and structural format. But then again going through the first few classes and I was enthralled by how we have built myths of what Software engineering means and what it is. Amateur programmers or engineers without experience can think in terms of

"If I I'll be able to complete 100% of the program that I am given, I am a good software engineer"

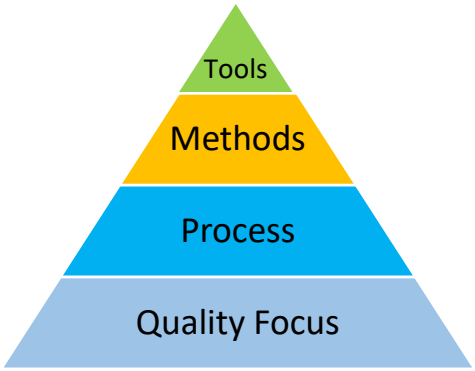
"Putting up all the documentation of tools that I have used from the tool's company website is enough for reference"

"Every time there is a new requirement, I can just make a new method to integrate that feature".

A bold no! As I was going through each and every class and chapter my mind broadened to actually and I feel that I am now confident to define what my role as a "Software Engineer" is or what it should be. Every software project starts with a business need. It requires us to solve a business problem using software, it could either be making an existing software adaptable to newer problem or make an entirely new product.

My understanding of What software engineering is a systematic integration of

Tools + Methods +Process + Quality



Software engineering was defined in many number of ways throughout time, but the wording itself is considered to be an aggregating of the 4 layers mentioned above in that very order.

I do not feel entitled to define what software engineering is myself, but I can define how I will tailor my roles and responsibilities for the Software Engineer designation in an IT industry. I arrived at this summation of four points by searching what the industry is expecting in a software engineer at entry level.

- Being well versed with one more modern-day programming languages, exceptionally well at understanding code, object-oriented programming analysis and design
- Applying engineering principles to solve complex problems through sound and creative design and engineering.
- Being able to learn new engineering methods and incorporate them into his or work processes'
- Ability to use various testing strategies, estimate risks and fix bugs.
- Seeks feedback and applies internal or industry practices to improve his or her technical solutions.
- Demonstrating skill in time management and completing software projects in a cooperative team environment.

Technology is run by corporate, be it gaming, music, health, business, entertainment. Software has become a key element for any technological product. This cannot be built by just a programmer, or an analyst a manager or a CEO themselves. A person who understands the technology's underlying concepts, plans, designs solutions and executes is in one line a software engineer and he or she has become a vital person in reshaping the society. Google today says there are close to 26 million software engineers as of today and expected to grow to till 29 million by 2024.

The engineer himself is responsible for highly expensive projects since a successfully implemented software can increase the business by innumerable amounts and a weakly implemented one could break an entire organization or put it out of business. One such example out of my experience in corporate is the recent Maze Cyber-attack that happened in a company in 2021 resulted in a 3 billion loss of clients to the company. Hence Software Engineer equals highly important and there is no negotiation. I understand the weight of this journey and cannot wait to get started. A software engineer himself can work can perform different kinds of roles that are required for a software project, he could be an analyst to study and review code, a tester who can increase the quality and efficiency of code, a manager who can allocate and divide tasks required for a project and so much more. To lay more emphasis on how important a software engineer is to an industry I would like to tell the story of a woman I heard in a TED talk who was the reason a world-renowned project that we know in the history of space flight was success. 'Margaret Hamilton', an alumnus of MIT, a software engineer in Draper's Lab during the apollo's mission in which Neil Armstrong landed on moon handled fixing a large error before the Apollo 11's Lunar landing.

About 3 minutes before the Apollo 11 was going to touch down the moon, warning lights started flashing, indicating that the Apollo Guidance Computer's CPU was overloaded. During the descent the astronauts toggled a switch for rendezvous radar in their checklist mistakenly which ended up flooding the CPU with large amounts of unnecessary data which was not required for the landing. Although this was an impossible task for Hamilton or the astronauts to find before the landing. Hamilton expected the situation of an overloaded CPU and wrote code that would clear low priority jobs from the CPU queue, flush memory, and restart CPU such that high priority

jobs can use space like the lunar landing module. Hamilton knew that she and her team had planned for this, and they had written code to handle this exact kind of problem.

The Software hence worked as it was required keeping Apollo 11's eagle lander on track until it touched down the moon's surface, if not the mission under Armstrong's command would have been called off and leading to a mission failure.

"Many of the things I was intrigued by had to do with how to make the mission software safe and reliable," -Hamilton

Hamilton's story also paved the name of a systemized practical approach to software programming to become software engineering later. So, can the industry work without software engineers? If Hamilton was not there, humans wouldn't have made it to moon.

Software engineering as of 2021 has become a diverse field with a requirement of vivid specializations in technologies like Internet of Things, Cloud Computing, Data Analytics, Data Science, Cyber Security etc. Hence there are increasing number of professionals in the number of computers to solve problems that run businesses. With the impact of pandemic, software engineers can work at ease from their home.

The increase in demand has also lead software engineers to start freelancing or start their own business for software engineering services for different problems. Taking the Hamilton's example as reference we can imply that software engineer is the go-to person for any implementation errors, bugs e.t.c. Hence, they are proficient in building high quality software programs.

Software engineering follows ethics like confidentiality, competence, intellectual property rights which maintain the proper business workflow of organizations. The job is interesting and challenging enough that there is never one approach to solve a problem.

But is it that easy? Software Engineering is undoubtedly a lucrative career that people could choose their work in however we often see a lot of people complaining about their job. Seems that there do exist cons of software engineering as a role and the study itself.

For example, post-pandemic, engineers although working from home, find themselves in an overlap of work-life balance. Out of personal experience, I have found engineers and developers working on more than two projects to meet deadlines and other business requirements. Engineers find themselves stuck in meetings for more than their scheduled times which has a detrimental effect on their productivity. Spending 8+ hours on a computer has negative effects on human bodies as there is mental work and less physical activity. But these are disadvantages faced by the engineers themselves there are fewer but a few cons in using the software engineering approach itself.

Too many rules, software engineering provides guidelines and principles on how to approach software projects and problems, but there are rules created and formed by various standards over time from organizations such as ISO, IEEE, ACM, etc. that engineers find hard to re-collect during the implementations.

Software Engineering is a practice and has a learning curve that must get better over time, entry-level engineers can find it hard or overwhelming trying to abide by those standards. Working too much on documentation can

make them feel unproductive. Corporate divides tasks into sub-tasks hence junior engineers could find it hard to see the bigger picture of their project.

Based on experience after contacting different software engineering practitioners since 2019 I understood how corporate today is using the engineering principles for projects. In class, during my undergraduate, I was introduced to different software project models for approaching a project. Be it the waterfall model, incremental model, the spiral model, etc. Most IT organizations have adopted the agile environment.

An Agile model is an iterative approach where requirements and solutions to the problems are self-organized between cross-functional teams to help businesses and customers deliver faster and quality results. Scrum is a sub-process that is a part of this model in which people follow a framework of activities to measure progress.

Scrum uses having daily scrum meetings of about 15 to 30 minutes, they timely have sprint retrospectives before reaching a milestone to discuss and analyze what has gone well and what didn't during the sprint. There exists a team manager and a scrum master in specific who perform the planning and tracking needed for the project. I encountered this approach in work and then during my masters when I understood that this is the methodology used by most of the companies.

Software Engineers apply the methodologies and processes learned as a part of this agile, for example making the documentations as the product implementation is evolving. When compared with the models learned during undergraduate, agile methodology feels like a perfect blend of spiral and incremental models with an additional lightweight scrum method. The Unified Modelling Language hence becomes a part of the implementation when the project is evolving.

It is seen that JIRA and Kanban are two renowned project management tools that are used in IT to schedule and keep track of the requirements, planning, and development that is to be taken place for every sprint and tasks that can be allocated to everyone in the team. The Kanban board is used for writing the works that are to be done today by the engineer, the concept is to mimic what happens in a "Hotel Kitchen", the orders which are to be served are written on a notepad, and assigned to the chef to finish. The sous chef and the head chef mimic the scrum master and the Project Manager.

The tasks in a software environment are divided into a to-do, In Progress, and Done. There is also a Backlog cart, which is to be finished by the engineer to pick up the next user story to work upon. Costs of the project can hence be calculated on the go with the backlogs left in addition to the time assigned for the current task.

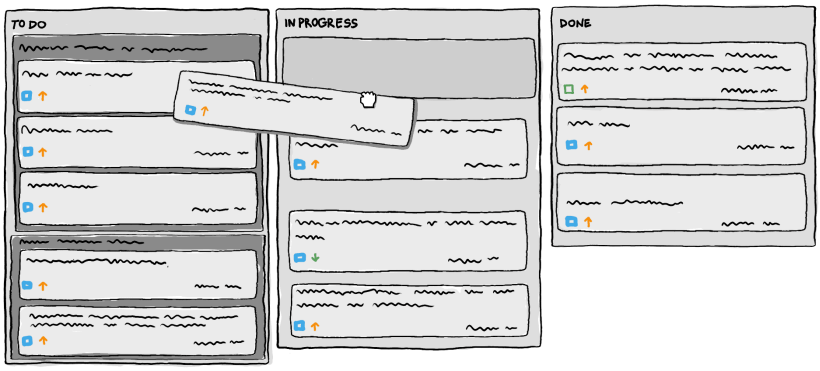


Fig. Illustrating use of a Kanban board to use Jira – (Comparing Agile to a Hotel Kitchen)

The common terminology used in implementing a feature is defined as a "User Story" (an evolved name for Use case) and most scrum projects undergo this phase of extreme programming (XP) between every sprint. It is made sure that every step taken is tracked and the defective or features that were not implemented properly are placed in a product backlog. The cost estimates hence are calculated with the current development that has to take place in addition to every backlog that has to be cleared. These user stories can be prioritized according to the importance of the requirements needed for the next sprint. This agile environment includes using a continuous integration testing deployment of the project to keep the business running. The environment serves as a major business solution for in-house solution software provided to clientele so that, the teams can be periodically changed in the company.

Hence the industry implements software engineering practices in various ways by using easier tools and methods that can increase the efficiency of products and business. As of today, there are further technologies like Trello, Wrike, etc that are used by companies are alternatives to Jira and Kanban boards.

These strategies have been referred to as Agile Modelling in Pressman's stressing an iterative approach of self-organizing, communicating, and working using extreme programming (XP). One interesting feature that is recommended while using an Agile environment is using "**Travel Light**".

Travel Light is a process of building or reusing those models that will only be used for a long term and dismissing rest. **Agile Unified Process** (AUP) is a philosophy that adopts **'serial in the large'** and **'iterative in the small'** meaning it abides to use a serial model like incremental or waterfall for a large software project whereas use an iterative approach like a spiral as a part of agile for a smaller to medium project. Another technical terminology that agile imparts in using projects is **Feature Driven Development (FDD**) which lets engineers concentrate on working on the important features (user stories) first and less important next.

With a study of requirements and preferred qualifications to the SE roles in the market today, almost each of them specified the line 'Should be able to work in an agile environment'.

Roles you imagine in as a software engineer in theory and in practice –

My theoretical perspective for a software engineer is a person who has strong fundamentals with design and engineering principles and can architect an approach for solving or implementing the given software problem in a given time or business constraints. In general, I imagined the role resembles that of a manager as the work sounds highly abstract and similar to management rather than being technical. But I found it to be so much more when I stepped into IT.

An old friend of mine works in FactSet, a medium-sized US-based limited software company where his designation is 'Software Engineer-1'. When compared about work between the work of the same role level in a multinational company versus a limited company FactSet, I've been enlightened of the fact why the IT folk and leaders like Jack Ma said, "If you want to learn work, join a startup whereas we could join a multinational company if we care too much about job security".

In a multinational company, an entry-level software engineer had to work to develop programs and submit them to the upper management similar to what happens in an undergraduate programming lab. A tester would do the testing work and just submit the work. Here, reaching the required objective is done, and no more or less than that happens until the person reaches higher role levels. But I was highly surprised how much difference exists in comparison to a small to medium-sized company.

My friend here in the medium-sized software company for reference works as a developer for the assigned user story in an agile environment and posts it to testing. Next, he is assigned to perform testing on the user story worked of another feature or project worked upon by his fellow teammate. After this, he gets to perform root cause analysis to the tickets (errors, bugs, or problems for customers) obtained for an in-house software product. Post this he gets to manage or assign subtasks pending to another in Jira/ Kanban board and finally, he gets to learn any one technology or certification as a part of his company's quarterly learning policies.

So, when looked closely an engineer himself gets to be a developer, tester, a support analyst, a scrum master, and finally a student. If asked, what roles do I imagine in a software engineer? My friend's example is what I aim to be.

Highly large companies, in general, have a specific corporate hierarchy stringently maintained as they make up a large part of the society. In such industries the usual roles in terms of software engineering are subdivided into the form Analyst Trainee < Analyst < Developer < Tester < Scrum Master < Project Manager < Architect < Delivery Lead < Product Managers and so on. Hiring for these roles is found to be specific hence the quality output is achieved since a person gets to perform fewer roles per designation than compared to small companies.

This software engineer hence not only learns the concepts of how to approach the problem but he or she will train themselves with the industry-specific (tools + methods + Process + quality standards). They learn whatever it takes to be a software engineer by complying with the organization's standards, tools, and processes. For example, consider a Use case to be drawn by a user story, our engineer is working on. The engineer as a student might have been trained to draw such diagrams using non-commercial software like StarUML but the organization he is working for needs him to draw using IBM Rational Rose.

IBM rational rose is a highly sophisticated well-built software used by organizations for project planning using UML. Learning to make a diagram with it takes more time than StarUML. Similarly with other technologies, processes, methods, etc.

So how does one know if they are doing the correct way of applying software engineering? Pressman has answered in the book saying we have to gain a firm understanding of the underlying concepts and choose to practice them with appropriate tools for the right task. It is successful if the recipe ordered is delivered reaches expectations in the right amount of time.

Not always the work of a software engineer is limited to the practical implementation of a software project. The foremost part of any project is 'Requirement Gathering' followed by planning. The term although self-exploratory there exists a formal division in the industry approach referred to as "Requirement Engineering".

Simply put the step involves researching, collecting, and allocating together the required resources of a project. All software projects start with a mere idea, textbooks term it to be **Inception** that involves asking the right questions needed for the project, this is followed by **elicitation** involves the actual gathering of requirements. Inception and Elicitation allow us to develop various use case scenarios from the clientele. Elaboration lets us develop the requirements needed and these are negotiated according to the talent pool available for a project with the stakeholders and clients to let them what in the project is possible and what is not? Which in turn allows to calculate of the project cost.

This primary phase of the project has been discussed to understand that research is an important phase of the project to be implemented. Industries allocate a 'Research and Development' division specific to this. Software Engineers in this division get to study the market, its trends, the kind of **product** that customers would love, the kind of **Price** they would be willing to pay, The **Place** in which the product would have the highest impact followed by the amount and kind of **People** – the target audience the product would have a high impact on. For example, A study in students of India indicates, they would choose a mediumly priced flagship variant of OnePlus that has the same performance as two overpriced branded companies like Apple or Samsung.

Researchers get involved at every phase of the project to recommend the right kind of practices that would suit a particular project. For example, The Architectural design needed for a certain project is decided by studying the architectural practices used which made a similar project successful.

Trending research areas in software are termed as 'Soft Trends' by Pressman and by referring vivid articles regarding the emerging areas in software engineer, it is understood that industries have started to tune or make developments to the engineering practices which are better suitable for respective technologies being used.

Few of those current-day technologies include Artificial Intelligence, Cloud Computing, Big Data, Deep Learning, Mobile computing, etc. One example of such change in the usual software engineering principles in the case of these new technologies is using automated software engineering methods that develop the code automatically when a respective UML diagram is made in the application. Robotic Process Automation (RPA) an emerging subdomain is a process that s used by companies that enables creating software bots that automate repetitive tasks like auditing and reviewing certifications once code has been pushed to production.

We are in the age where Pressman has mentioned in his 'Software Engineering: A Practitioner's approach' as the 'Grand Challenge' for Software Engineers which involves making software which in turn make it easy to make softwares. Azure for example is provisioning the power of cloud computing using the highest security protocols involving the cyber security realm. Hence the practices develop and tune according to the change in technology although the underlying principles and guidelines remain the same.

Advanced Software Engineering has been a fun ride filled with mixed experiences. The course has given me a deep dive into how IT implements the engineering practices and justified applying to our final projects. Initially, I learned the fundamental principles and the process models in Undergrad but when I worked as a programmer analyst, I was working in an agile environment, but I failed to recognize it due to lack of knowledge.

The Daily Stand up, reaching milestones, Developers using Jira, Trello or Kanban Boards, etc and I didn't know. My work was entirely providing support to the in-house solutions created by the developers but I failed to see unless I was taught about agile methodologies in class. At the beginning of the course, my mere thought of software engineering was to prepare a set of software process models in which software projects are executed such as waterfall model, incremental model, spiral model, and UML diagrams.

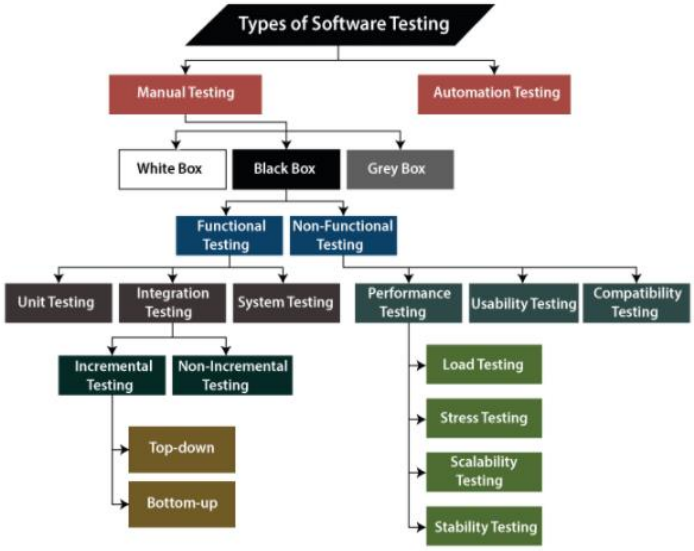
But three weeks far I learned a practical approach for every chapter I have encountered. Putting aside the definitions and UML concepts, I felt creating a project proposal document was very interesting. I got to study the market analysis and deduce what are the similar kinds of applications that are successful and how can we be different. So, studying the market helped us gain the skills needed to sell the project. The unique selling point of our project is to target and make the conventional bus travel easy for the disabled commuters of Windsor.

Followed by this, I learned to approach a Software Specifications Requirements (SRS) document. This document helped the team to undergo a mere idea journey to inception, elicitation, collaboration, negotiation and then reaching the set of requirements that will be implemented in the project.

Creating UML diagrams was a fun task, but when implementing the coding phase of a project we understood the importance of design patterns that are helpful during the coding phase of a software project. Software engineering design patterns using GRASP methodologies and Gang of Four (GOF) patterns are helpful in converting the required project infrastructure into a working code. Not all design patterns could be drawn with the given time constraints however their rough sketches and methods were helpful while trying to code. For example, using the Adaptor patterns while implementing the UI of the project.

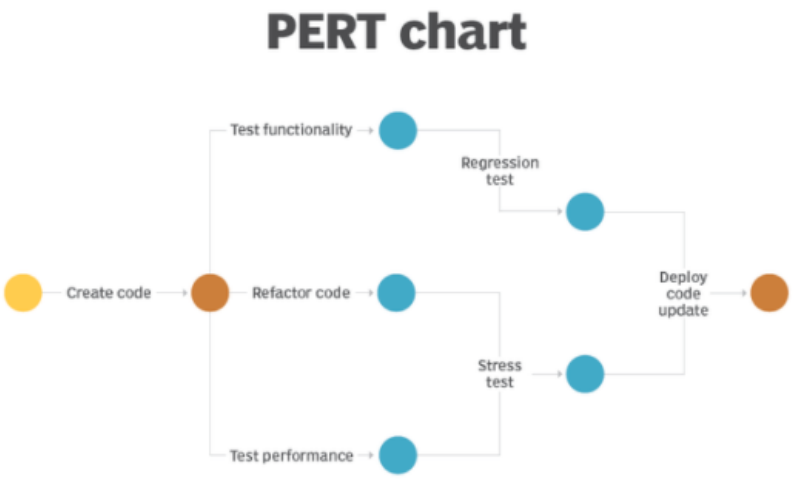
The designed course was helpful in recognizing the power of software engineering in a corporate workplace and gave us the ability to work in a agile workspace which is a high requirement skill preferred in the IT today. We practiced being a part of an agile environment by tightly abiding by the daily scrum meetings, conducting a sprint retrospective, reaching milestones. This approach helped in delivering quality results in limited time constraints when compared to approaching a single-handed capstone academic project.

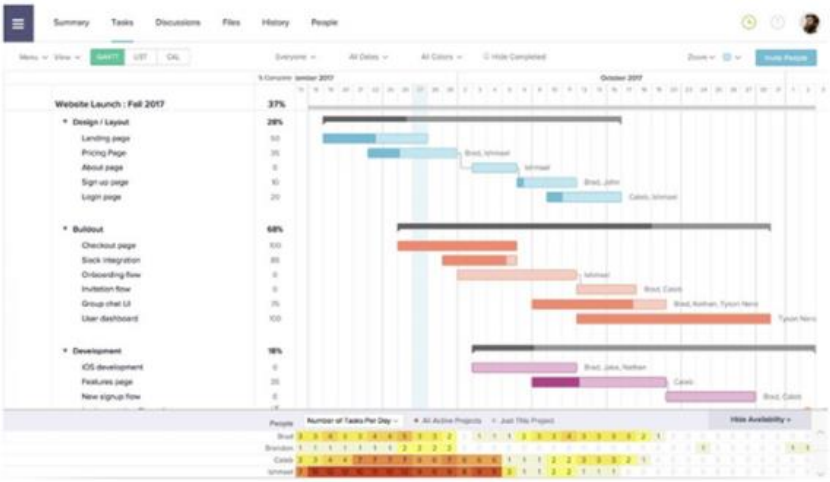
Using Agile tools like Jira, Kanban and Github were practiced to gain the crux of agile methodologies. The next interesting phase of the subject was to encounter various testing methodologies. I studied various testing methodologies only in theory, but I didn't get to use them in reality. The course helped me gain knowledge on 20 plus testing strategies that can be used on software projects.



Due to the available time constraints, it was hard gaining to practice all possible testing strategies and scenarios but thinking in terms of testing made me a better developer since we estimated various kinds of possibilities a running program could face and handle them, thus make us understand the importance of well-crafted code.

Last but not the least, we understood calculating the Risks. Risk Management is a sub domain highly required in IT and needed estimate the risks that we could encounter and give an insight of how those risks can be prevented in software projects. In Risk management we also learned how product metrics are needed to analyse the cost of project at every stage. Finally, using visualization tools such as Pert and Gant charts to calculate the project costs.





Gantt Chart (via TeamGantt)

10 weeks ago, I was a person who only knew the technologies and the required soft skills needed to crack a software engineer interview. But I feel I gained the fundamentals needed to ace as a software engineer himself given an opportunity in such environment. The course Advanced Software Engineering provided an insight of what could be the life of a software engineer whilst making us practice being a software engineer every over the given weeks.

But I will also suggest a few betterments to the course whilst justifying why those changes could increase the efficacy of such an advanced course.

Becoming a full stack developer in MANG companies is a dream I would like to see myself making in the near future. As a full stack developer, I get to create integrate and maintain web applications. I would like to apply the concepts I have learned from this course, to approach my academic projects using the industry way by using the software engineering guidelines and principles.

My enthusiasm towards provisioning applications using the power of cloud computing has increased leading towards learning about current day cloud services like Azure and AWS. I plan on finishing the Azure fundamentals and later on set up a training path towards Azure Full stack development program.

My personality as of now matches the profile of a programmer analyst or a support engineer for an in-house developed web solution. I am learning different technologies like React, MongoDB, git along with agile technologies like Jira and Kanban boards to tune myself for the developer roles of a software engineer.

I will look forward for software engineer opportunities be it small to medium companies at first to learn being adaptable to the worst of conditions, so I can be proficient to a broad perspective of jobs in bigger companies such as Microsoft, Amazon and Delloite.

The course gave us an opportunity to bridge the gap between today's industry standards and the software engineering guidelines. I feel more design patterns could be given an insight to approach coding for software developers easy. The course missed the part where it has to give us a chance to teach and learn a new technology as it focused to use the technologies that students already used in their previous experience at industries or academics.

For example, teaching the use of Jenkins for continuous integration and maintenance of softwares. Laying emphasis on learning one new technology either for development or testing could have made the course further intense and useful. Another opinion is the amount of focus that could be paid, it is harder for student to pay focus for a continued period of 3 hours straight and could lead them to missing a few contents taught. Other than the above-mentioned details, Advanced software engineering was a fun filled ride which helped us gain the crux of transforming ourselves into better software engineers for the future.

THE END