Guest Lecture Summary Report

Topic: Software Development & Evolution at [redacted]
Speaker: [redacted]
Lecture Date: [redacted]
Prepared by: [redacted]
Student ID: [redacted]

This week we had the last guest lecture exploring how [redacted] use the principles of software deployment and evolution in their workplace. The major theme in this week's talk revolved around providing a direct link to the unit and its use in a real life setting. [redacted] first provided a detailed overview about him and the company, noting that he has been in the industry for over 10 years and currently works in a blended developer, DBA, trainer role. [redacted] currently support the 5th largest trading platform in the UK with billions of funds under management. Hi organisations work to support this application was used throughout the talk to link everyday roles to the impact it has on such a real world application. Towards the end, [redacted] opened up the floor to a question & answer session, allowing for specific questions to be answered.

[redacted] began by explaining what software deployment is, stressing the point that it is not the final step in the process anymore. It forms part of the iterative software development and maintenance process. Furthermore, he added, you are not only deploying to end users, but constantly deploying to other developers as well. [redacted] then explained what DevOps is in his view, really linking his view closely with that of the first guest lecture from [redacted]. [redacted] explained that it is important to deploy with repeatable and reliable processes that allow for amplification of feedback loops. He also stated that its important to develop and test against **"production like"** system to avoid the all too common 'works on my system' excuse. He stated that while the development environment is generally nowhere near as powerful as production system, in his case, having potentially half a TB of memory, we can still code for this to provide a similar (not identical) environment. He briefly mentioned this is the link to infrastructure as code.

Before moving on to [redacted] own practices, [redacted] explained that the goal of all these processes, e.g. Agile, CI, CD, DevOps etc. is to embrace and manage inevitable change. Fundamental to this is team work and communication that occurs through sharing of responsibilities, success and failure as well. [redacted] also stated that while big companies like Amazon have 100 or 1000s of deployments daily, his company would be lucky to do 1 a week. Instead their focus in on lots of testing due to the nature of the software. He recommends that each function should have a separate test case which should fail if the behaviour of the function has been changed in any way. This gives the developers a lot of confidence.

This was his lead in to explaining [redacted] practice, using the trading platform has his go to example as its their biggest project. The software is 25 years old and considered a legacy system. However, many of the legacy features such as *Foxpro* databases have been moved to *Postgres*, which had to be done as the database size has grown from about 20GB to 800GB. They also have 12 small teams of 3-6 each working on about 55 software projects at any time. Their largest repo has 4 million lines of code and their primary goal is to carry out monthly releases which include 90-120 change requests. Furthermore, he stated that each update will cause a schema change and this needs to be updated on all 8 database servers they have, including connections to 7 application servers. All these numbers really help grasp the magnitude of the task and the need for automation and repeatability which he stressed over and over. His signature piece of advice that I took away was **"Don't let the idea of growing your system scare you"**. [redacted] also stated that his company's goals are to ensure that code is always in a deployable state and automate as much as possible in order to make deployment efficient and allow for better monitoring of systems.

A major component of his talk involved walking us through the typical workflow for a change request at [redacted]. These are usually done in the following order: Identify change requests. Review change requests as people done really know what they want. Develop the change request followed by a series of integration testing and user acceptance testing. Finally, the change request is put into production and patches are done if required. This process gets repeated over and over. While this is the typical flow, it is not always followed. The reasoning behind this is that it allows for iteration throughout the workflow and enables plenty of review and validation along the way.

The priority for [redacted]'s team is to ensure the product is of high quality. He suggests developers to not jump straight into writing code and have a good think before starting and conduct design and code reviews regularly. He raised the concept of technical debt, which really means if you take a shortcut remember to pay it back (by fixing it or improving the system in another way later; but not too late!)

[redacted] is a strong advocate for automation as mentioned before, and he raised this again, in the context of ensuring that you have repeatable processes which ensure the production deployment has no surprises. This also complements the idea of monitoring your system and code regularly as it allows the developer to catch production issues and react ASAP.

To finish up this section [redacted] stated you must deal with reality. That is that good practices prevent issues and allow you to identify them as early as possible. However, we must know how to deal with imperfection as sometimes you will need to pull change requests out of a release and roll back production. He also stated that its important to have processes in place for unplanned events, giving the example of the recent 'Brexit' that occurred in UK.

[redacted] finished his talk by briefly mentioning some of the tools that he uses at [redacted]. These include Git for source control, Jira for issue management, SonarQube for code quality management, Salt & Vagrant for virtualisation. He also mentioned various in house tools used to update databases using a single click, as well as monitoring, continuous integration and testing. In addition to these he also mentioned some less familiar tools such as *Rundeck* for deployment to multiple servers, *PgBadger* to analyse Postgres SQL performance, Logstash to collect and analyse logs and *Nagios* for overall infrastructure monitoring.

Relevance to ULOs

1. **Describe the activities in software deployment and apply these in a problem context.**
   This has been addressed as [redacted] took us through the common tasks involved in software deployment at his workplace, in relation to their major project.
2. **Apply knowledge of software environments to plan development and deployment.**
   [redacted] walked us through the software environments he works with and the processes he uses to plan development and deployment. He also stressed the need to automate deployment so that there is less variability in environment variables.
3. **Describe and classify issues that drive software maintenance.**
   This was a common theme, addressing the common 'works on my system' excuse as well as how to ensure high quality by increasing testing and using various principles.
4. **Describe latest practices in software development deployment and evolution.**
   He explained the various personally unheard of tools such as Rundeck and pgBadger used at [redacted] and the workflow for a change request in great detail.
5. **Draw on industry practice to evaluate the maintainability and design consistency of a software system.**
   Similar to ULO 4, many of his suggestions revolved around automating as much as possible in order to deliver repeatability and consistency. He also stated that maintenance is vital and ongoing part of the process, and so is change. We must learn to deal with both and provided tips on how to do this as best as possible.