Lecturecast Week 3: Software Development Life Cycles

Software development life cycles can be divided into two categories of approach:

1) Adaptive: Scrum, Kanban, Test driven development (TDD), Behavioural driven development (BDD) are all forms of adaptive approach. In adaptive approach, the phases of the SDLC can be repeated multiple times to get the desired result.
2) Predictive

Stages of SDLC

1) Collecting requirements
   The Volere template can be used to collect requirements. It consists of various sections that help the developer gain full understanding of the important aspects of the software development process. Functional and non-functional requirements are important aspects of the Volere template. The functional requirements captures what the system will do and the non-functional requirements captures how the system will do it.

   Non-functional requirements include:
   - Look and feel requirements
   - Usability and humanity requirements
   - Maintainability and Support requirements
   - Performance
   - Operational and Environmental requirements
   - Cultural requirements
   - Security requirements
   - Compliance requirements

2) Managing customer expectations
   The three constraints quality, time and cost need to be carefully managed to ensure that customers' expectations are met. If a customer wants a desired quality, they need to understand the time and cost constraints involved in achieving that quality and these should be documented and agreed by all stakeholders.

3) Analysis of requirements: Requirements must satisfy the SMART criteria.
   S- Specific
   M- Measureable
   A- Attainable

   R- Relevant

   T- Time bound

4) Design

Existing architectures and design patterns can be explored at this stage. Tried and tested solutions help to speed up development process.

5) Development
   Version control or source control can be used here to help track and manage changes to software code. If a mistake occurs, then it is possible to quickly trace the point of mistake when developers back track to a previous version.

6) Testing
   Testing can be done in stages:
   Unit testing: testing individual components of the software to check if each one is fully functional.
   Integration testing: individual units are tested as a group to identify interface defects.
   System testing: the software is tested as a whole to ensure its compatibility across the entire system.
   Acceptance testing: this is done to access if the system is ready for release.

Test Driven Development (TDD)

This approach to software development requires the software requirements being converted to test cases prior to development. The software is then tested against each test cases as development progresses. Testing occurs from start of the development phase till the end.

Behavior Driven Development (BDD)

This approach is focused on ensuring that developers understand what customers want and that customers also understand what is technically possible. It is based on an example-driven approach where examples are provided to describe how an application is supposed to work. BDD uses a plain English syntax know as Gherkin to capture test cases which removes any ambiguity that might have been introduced from prototypes.

Project Closure

These four steps should be followed for effective project closure:

1) Verify final delivery is complete: this involves checking that customer has received the final product are happy with delivery.
2) Hold a closeout meeting: share lessons learned amongst project team. Identify areas of improvement.
3) Release team and materials for other work: letting the team know they can move on to other projects and also taking the opportunity to provide feedback to their managers.
4) Ensure project documents are properly filed.

<u>Mistakes made during Software Engineering</u>

Ghazi et al (1991) discussed some mistakes made during software engineering:

1) Coding is started too soon.
2) Testing is unsystematic and/or insufficient.
3) Requirement and quality features not determined.
4) Standards and Regulations are ignored.
5) Missing, outdated insufficient or inadequate information.
6) Unrealistic deadlines.

Learnings from Video 1

Title: Project Closure - How to write a Post-Mortem Report

Link: https://www.youtube.com/watch?v=EJbWAeERakc&t=3s

A post-mortem is a process that is conducted at the project closure to analyze and access what worked well and what did not.

Best practices for conducting a post-mortem

1) Wait to conduct the post-mortem after the fire stops burning. It gives the team enough time to gather their thoughts
2) Limit the post-mortem session to 90 minutes
3) Create a proper structure which covers the following:
    - Common understanding
    - Individual roles
    - What worked well
    - What could have been done better
    - Most important lessons learned
4) Include only the right amount of people in the room – Only those that can benefit from the meeting.
5) Publish notes from the meeting including next steps.

Post mortem can also be done at end of individual project phases.

Learnings from Video 2

Title: Conducting post-mortem meetings without blame

Link: https://youtu.be/OJ9g44ggTTE

Tips for conducting a Post-mortem meeting

- Get an Agenda
- Write a detailed documentation that covers all the things that well and didn't go well. Remember to include your client's input too
- Structure your post mortem meeting as follows:

- Begin with objectives of the project (the purpose of the project, expectations, project parameters and cost) in order to ensure everyone is on the same page.
- Ensure to keep everyone focused (Avoid blame games)
- Ask leading questions.
- Don’t get defensive.
- Publish the report for everybody to learn.

Video 2 emphasises

- The importance of avoiding blame games at post-mortem meetings so all parties can get the best of lessons learnt.
- The need to produce a transparent, honest and detailed documentation.
- The actions of the project manager which include taking responsibility where necessary and focusing on lessons learnt.