Manjula Nannuri
Day-2_ Assignment-2

Assignment 1: SDLC Overview - Create a one-page infographic that outlines the SDLC phases (Requirements, Design, Implementation, Testing, Deployment), highlighting the importance of each phase and how they interconnect.

Requirements: ements in detail.

The purpose of this meeting is to more thoroughly understand the client's wants and needs, including a description of the software, who the end user will be, and its overall purpose. This information is then written into the software requirement specification (SRS) document.

Once the team creates the SRS document, they pass it to the client for approval. The SRS document can then serve as a guide throughout the designing and development processes.

Designing: During the design phase, the project team develops a working software prototype based on the desired features and user requirements.

The design process typically requires designers to create and test several design elements and ideas before selecting the final prototyp Requirement gathering involves collecting all relevant information from the client and using it to create the product, ensuring their expectations are met. To do so, business analysts and project managers meet with the client to discuss software require. The implication of the design to the entire software development process is that it guides the developers to create a working representation of the client's expectations while considering user-friendliness and multiscreen compatibility.

For example, if a team is developing an online shop, the design phase might consider the back-end framework, shopping cart features, payment features, and user experience parameters. They might also consider how the checkout page should display to the customer.

Implementation: The implementation phase is where the design created in the design phase is implemented into the necessary application programming interfaces (APIs) and project components. The result of this phase is a fully functional product.

After implementation, the final product requires further testing to address all bugs and discrepancies before release. The team measures the software against the specifications in the SRS document and sends a test version to reviewers. The feedback gathered during this phase allows developers to make necessary product changes before full implementation.

Testing: In the testing phase, developers—especially DevOps professionals—verify the software meets the predetermined requirements, designs, and other quality standards. Without proper software testing, the system may contain bugs or vulnerabilities that can go unnoticed and potentially lead to serious problems when put into use.

An example of this is deploying an application to a live environment like the Google Play Store. The application must be tested on various screens beforehand to ensure it works as intended. This includes validating data input, measuring application performance, and confirming security features are up to parameters.

Deployment: The deployment phase typically consists of putting the software into a production environment so it's accessible to users. After successful deployment, customers can use the newly developed software. In some cases, the client may request user acceptance testing before deployment to ensure the software meets expectations.User acceptance testing involves testing the newly created software and ensuring it performs correctly and meets specific requirements.

Assignment 2: Develop a case study analyzing the implementation of SDLC phases in a real-world engineering project. Evaluate how Requirement Gathering, Design, Implementation, Testing, Deployment, and Maintenance contribute to project outcomes.

Requirements: Requirement gathering involves collecting all relevant information from the client and using it to create the product, ensuring their expectations are met. To do so, business analysts and project managers meet with the client to discuss software requirements in detail.

The purpose of this meeting is to more thoroughly understand the client's wants and needs, including a description of the software, who the end user will be, and its overall purpose. This information is then written into the software requirement specification (SRS) document.

Once the team creates the SRS document, they pass it to the client for approval. The SRS document can then serve as a guide throughout the designing and development processes.

OverView: The SRS documents that defined the project's specific objectives and needs will then be used as a point of reference for the next stage.

The SRS will need to include all the software, hardware, security, and specifications needed.

Planning: Once gathering requirements is completed, the next step is to plan. From there, the terms of the project will be determined:

- Cost estimation
- Required resources
- Tech stack
- Roles needed in the project
- Project scheduling
- Leadership structure

The person mainly in charge: Team Leaders and Higher Management

Project managers will need to clearly outline the project's scope and objective. This charts the team's trajectory to efficiently develop the software. Additionally, it establishes constraints to prevent the project from escalating or deviating from its initial objective.

OverView: An expanded SRS document with product feasibility analysis and risk identification.

Based on this outcome, you can define the appropriate technical methods that should be implemented for the project to run smoothly with minimum risk

Designing: The third phase goes into great detail about the required standards, features, and activities that fulfil the functional specifications of the proposed system.

Some essential factors that you need to take into account:

- Architecture: quality attributes, IT environment, human dynamics, and business strategy.
- User Interface: how users interact with the software.
- Platforms - decide on which platform the software will launch (Windows, Android, iOS, and Macintosh).
- Programming: involves techniques of resolving issues and executing tasks in the app alongside the programming language.
- Communications: specifies the channels through which the software can talk to other assets, e.g. a central server.
- Security: describes the procedures taken to safeguard the program, which might consist of password protection, SSL traffic encryption, and secure user credential storage.

During this SDLC step, a prototype is worth considering. Although it is time-consuming, prototyping will be much more affordable than making significant modifications after the software development phase.

The person mainly in charge: Architects and Senior Developers

A person who is responsible for this will gather this info in the Design Document Specification (DDS). Stakeholders then will examine the DDS based on the software's risk assessment, design modularity and time constraints.

OverView: A detailed DDS that lists all the information required to code the product.

Implementation: The implementation phase is where the design created in the design phase is implemented into the necessary application programming interfaces (APIs) and project components. The result of this phase is a fully functional product.

After implementation, the final product requires further testing to address all bugs and discrepancies before release. The team measures the software against the specifications in the SRS document and sends a test version to reviewers. The feedback gathered during this phase allows developers to make necessary product changes before full implementation.

OverView: Source code of a testable, fully functional software.

Testing: This stage of SDLC helps ensure the software has no bugs or exploits and can run smoothly.

Businesses have various testing strategies to evaluate the new product, including

- Performance testing
- Security testing
- Unit testing (functional tests)
- Automation testing
- Integration testing.

When the product is stable, bug-free, and up to the quality criteria outlined in the earlier stages, the testing step is complete.

The person mainly in charge: Quality Assurance (QA) specialists.

QA engineers usually deploy a range of frameworks in addition to continuous integration to perform unit tests and automation compilation. Testers will need to have a combination of manual and automated testing knowledge and skills to perform the tests effectively,

Output: A thoroughly tested version of the product ready for a production environment

Deployment: The deployment phase typically consists of putting the software into a production environment so it's accessible to users. After successful deployment, customers can use the newly developed software. In some cases, the client may request user acceptance testing before deployment to ensure the software meets expectations.User acceptance testing involves testing the newly created software and ensuring it performs correctly and meets specific requirements.

The person mainly in charge: Project Managers

OverView: The release of a fully functional and tested product

Conclusion: Having an effective SDLC is vital to develop a high-quality software solution. Failure in one stage can have negative effects on the remaining stages of the cycle, hence, jeopardizing the success of the project. Businesses can only take advantage of SDLC when the plan is meticulously followed.

Want to design an efficient SDLC that can streamline the software development cycle and create transformative products? Consult with our experts today to kick-start your project and have professionals accompany you through every stage of the SDLC.

Assignment 3: Research and compare SDLC models suitable for engineering projects. Present findings on Waterfall, Agile, Spiral, and V-Model approaches, emphasizing their advantages, disadvantages, and applicability in different engineering contexts.

1. Waterfall Model:

Advantages:

• Sequential Process: Progresses linearly through phases (requirements, design, implementation, testing, deployment).

• Clear Milestones: Well-defined stages with specific deliverables at each phase.

• Documentation: Emphasizes thorough documentation, facilitating clarity and traceability.

Disadvantages:

• Rigidity: Little flexibility for changes once a phase is completed.

• Late Feedback: Stakeholder feedback often comes late in the process, increasing the risk of costly changes.

• Long Delivery Time: Large projects may experience delays due to sequential nature.

Applicability: Suitable for projects with well-understood and stable requirements, where

changes are unlikely or can be managed through formal change control processes.

2. Agile Model:

Advantages:

• Flexibility: Embraces change and allows for iterative development.

• Customer Collaboration: Regular feedback from stakeholders ensures alignment with

customer needs.

• Early Delivery: Prioritizes delivering working software in short iterations.

Disadvantages:

• Complexity: Requires highly collaborative and self-organizing teams.

• Documentation: May lack comprehensive documentation, leading to knowledge gaps.

• Scope Creep: Continuous changes may lead to scope creep if not managed effectively.

Applicability: Ideal for projects with evolving requirements, where frequent delivery of usable

increments is valued, and customer involvement is crucial throughout the development

process.

3. Spiral Model:

Advantages:

• Risk Management: Emphasizes risk identification and mitigation throughout the project

lifecycle.

• Flexibility: Allows for iterative development and refinement of prototypes.

• Feedback Loop: Incorporates customer feedback early and often, reducing the risk of

late-stage changes.

Disadvantages:

• Complexity: Requires thorough risk analysis and management expertise.

• Resource Intensive: Prototyping and iterations can consume time and resources.

• Suitability: May not be suitable for small or straightforward projects due to its complexity.

Applicability: Well-suited for projects with high uncertainty and evolving requirements, where early identification and mitigation of risks are critical.

4. V-Model:

Advantages:

• Traceability: Emphasizes traceability between requirements and test cases.

• Early Testing: Testing activities are initiated early in the development lifecycle.

• Structured Approach: Provides a structured approach to software development and testing.

Disadvantages:

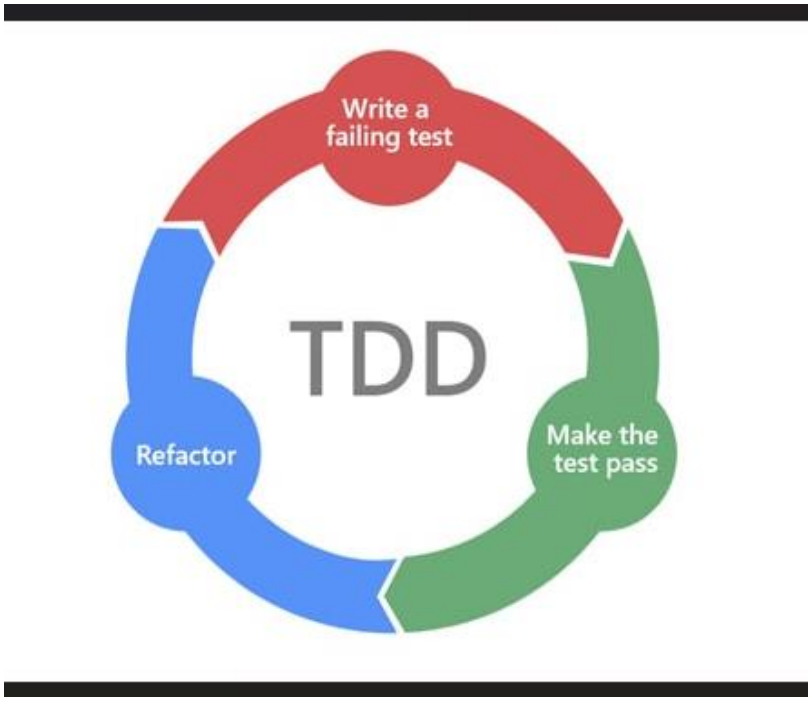
• Rigidity: Similar to Waterfall, changes late in the process can be costly and time-consuming.

• Documentation Overload: Requires extensive documentation, which can be cumbersome.

• Sequential Nature: Limited flexibility for iterative development and customer feedback.

Applicability: Suitable for projects with clear and stable requirements, where thorough testing and traceability are critical, and changes are minimal or well-controlled.

Assignment 4: Create an infographic illustrating the Test-Driven Development (TDD) process. Highlight steps like writing tests before code, benefits such as bug reduction, and how it fosters software reliability.



Test-Driven Development (TDD) Process Infographic:

1. Write Test:

Developer writes the test case without any piece of code in front of him. This is mundane and time-consuming, but make sure that the developer understands the functionality essential and in accordance with that write the test case. The test case is not biased to showpiece of code works rather test cases are to test the expected functions.

2. Run Test:

The preceding step that follows is to ensure the correct working and that the new test doesn't pass by any sort of mistake without any new code. This step also eliminates the possibility that the new test always passes.

3. Write Code:

In the next step writing code is done that clears the test. The new code isn't perfect but is later changed as per the necessities. It's merely designed to test and doesn't enclose other functionalities. Focus is on writing only what's needed to fulfill the test requirements.

4. Run Test Again:

If every single test case produced simply passes the test, it implies that the code meets all needed specifications. Hence the next step of the cycle can be Refactor the code to remove duplication and to clean it

5. Refactor your Code:

This is exactly the same as removing duplication. A refactoring doesn't damage any existing functionality and assists to remove duplication between production and test codes. The code is now cleaned as needed.

Assignment 5: Produce a comparative infographic of TDD, BDD, and FDD methodologies. Illustrate their unique approaches, benefits, and suitability for different software development contexts. Use visuals to enhance understanding.

1. Test-Driven Development (TDD):

Approach:

TDD is an agile development methodology where tests are written before the code is developed. In contrast, traditional testing is performed after the code is written.

Benefits:

Ensures software is built with testing in mind from the beginning, leading to fewer bugs and higher quality.
Encourages simple designs and modular code as developers focus on writing code to satisfy specific test cases.
Provides a safety net for refactoring, allowing developers

2. Behavior-Driven Development (BDD):

Approach:

Focuses on behavior and outcomes rather than implementation details.
Uses natural language specifications (e.g., Given-When-Then) to define tests.

Benefits:

Facilitates communication and collaboration between developers, QA teams, and non-technical stakeholders by using a common language.

Helps ensure the development process aligns with business goals and user expectations.

Encourages a test-first approach similar to TDD but with a focus on behavioral specifications rather than low-level unit tests.

Suitability:

BDD may not be suitable for short development cycles or projects with frequently changing requirements.

3. Feature-Driven Development (FDD):

Approach:

Focuses on building features incrementally based on client priorities.

Emphasizes short iterations and frequent client feedback.

Benefits:

Incremental Delivery: Delivers tangible results to clients in short cycles.

Client-Centric: Aligns development efforts with client priorities and business

objectives.

Scalable: Scales well for large, complex projects with multiple teams.

Suitability:

Suitable for large-scale projects with evolving requirements and multiple stakeholders.

Best suited for projects where client involvement and feedback are essential.

Conclusion: Each methodology offers a unique approach to software development, catering

to different project requirements and team dynamics. Whether it's the test-driven approach of

TDD, the collaborative nature of BDD, or the feature-centric approach of FDD, choosing the

right methodology depends on factors such as project size, complexity, and stakeholder