CSI3025

Application Development and Deployment Architecture

Winter Semester 2023 – '24

Digital Assignment - 1

SARAGA S
20MIC0081

Digital Assignment - 1

CSI3025 - Application Development and deployment Architecture

SARAGA. S
20MICOOGI
Slot: G1+TG1

1) Agile model is a dynamic model - Justify.

Agile model is considered a dynamic model primarily because of its iterative and incremental approach to software development. Here are some justifications:

* Iterative development: Agile emphasizes breaking down the software development process into small, manageable iterations called sprints. Each iterations results in a potentially shippable product increment. This iterative approach allows for continuous feedback and improvements throughout the development process.

* Flexibility and adaptability: Agile methodologies such as Scrum prioritize adaptability to changing requirements. The dynamic nature of business environments often leads to changes in priorities and requirements. Agile methodologies are designed to changes in priorities and requirements. Agile methodologies are designed to accomodate and embrace these changes, allowing the project to adapt quickly to evolving circumstances.

* Customer collaboration: Agile places a strong emphasis on customer collaboration and continuous customer feedback. By involving customers in the development process, teams can respond to changing customer needs and preferences, ensuring that the delivered product better aligns with the customer's expectations.

* Continuous Planning and monitoring: Agile involves continuous planning, ~~process allows~~ monitoring, reassessment of project goals and priorities. This dynamic planning process allow teams to adjust their strategies and priorities based on feedback, changing market conditions, or evolving project requirements.

* Incremental deliveries: Agile promotes the delivery of a working product incrementally and regularly throughout the development process. This incremental delivery allows stakeholders to see tangible progress at regular intervals and provides the opportunity for early identification and correction of issues.

* Emphasis on Individual and Interactions: The Agile manifesto values individuals and interacting over processes and tools. This fosters a collaborative and communicative environment where team members can respond quickly to challenges and circumstances.

* Continuous Improvement : Agile encourages a culture of continuous improvement. At the end of each iteration, teams conduct retrospective meetings to reflect on what went well, what could be improved, and how to enhance their processes. This commitment to continuous improvement contributes to the dynamic nature of Agile development.

2) Assume that you are the technical manager of a software development organization. A client approached you for a software solution. The problems stated by the client have uncertainities which lead to loss if it not planned and solved. Which software development model you will suggest for this project - justify. Explain the model with its pros and cons.

The spiral model is particularly well-suited for projects with high uncertainty and the need for frequent risk assessment and mitigation.

Spiral model :-

* Risk management : Spiral model is iterative and involves a series of repeating spirals, each representing a phase in the software development process. During each spiral, risks are identified, analyzed and mitigated. This iterative risk management approach is beneficial when dealing with uncertainities because it allows the team to address and adapt to changing requirements and risks throughout the project.

* Flexibility : The spiral model provides flexibility in incorporating changes as the project progresses. This is crucial when the client's requirements are not well-defined initially, as it allows for adjustments based on evolving needs.

* Incremental Development : Like other iterative models, the spiral model supports incremental development. This means that the project is built incrementally in smaller, manageable parts, allowing the client to receive partial implementations early in the development process. This helps in early validation and ensures that the project is heading in the right direction.

* Client involvements This model allows for significant client involvement and feedback throughout the development process. Regular prototypes and versions are delivered, fostering better communication between the development team and the client. This ensures that the delivered product aligns more closely with the client's expectations.

Pros:-
* Risk management
* Flexibility and Adaptability
* Client involvement
* Early prototypes.

Cons:-

* Complexity : This model can be complex to manage and requires experienced project management to handle the iterative and risk-driven nature.

* Time and cost : The iterative nature may lead to an extended timeline and potentially higher costs.

* Documentation :- Due to the iterative nature, extensive documentation may be required at each spiral, adding to the project overhead.

3) What do you understand about the build? How it is important in application development? What are the processes followed by a software development team when building the software?

In software development, a "build" refers to the process of compiling and linking source code files and other resources to create an executable or deployable version of a software application. The build is a crucial step in the software development lifecycle as it transforms human-readable source code into machine-executable code that can be run on a computer. The build process encompasses various tasks such as compilation, linking, packaging and sometimes testing, depending on the development and release requirements

Importance of Build in Application development.

* Error detection: The build process helps in detecting syntax errors, compilation errors, and other issues early in the development cycle. This allows developers to address and fix problems before moving to subsequent stages, reducing the likelihood of defects in the final product.

* Integration Testing: Builds are essential for integration testing, where different components or modules of the software are combined and tested as a whole. Integration testing during the build process ensures that individual units work cohesively when integrated.

* Deployment: The build process is a precursor to deployment. It produces the executable or deployable artifacts that can be installed on target environments for testing or production use.

* Consistency: Builds ensure consistency across the development and testing environments. A standardized build process helps in creating reproducible and predictable results, reducing compatibility issues between different environments.

* Automation: Automating the build process enhances efficiency and reduces manual errors. Continuous Integration (CI) and continuous Delivery (CD) practices rely heavily on automated build processes to ensure a steady and reliable flow of changes into production.

Processes followed by a Software development team during Build:

* Compilation: Source code files are translated into machine-readable code by a compiler. This step checks for syntax errors and ensures that the code adhere to the programming language rules.

* Linking: The compiled code and necessary libraries are linked together to create executable files or libraries. This step resolves references between different code modules.

* Packaging: The compiled and linked code, along with other required resources (configurations, assets, etc), is packaged into deployable units. This could be an installer, a package for a package manager, or other distribution formats.

* Testing: In some cases, testing is integrated into the build process. Automated tests can be executed to ensure that the build meets quality standards. This can include unit tests, integration tests, and even acceptance tests.

* Versioning: The build process may involve assigning version numbers to the software artifacts to track changes and releases systematically.

* Documentation: Generating documentation, such as release notes or user manuals, can be part of the build process to ensure that users and stakeholders are informed about changes and features.

* Deployment: In some cases, especially with continuous deployment practices, the build process might trigger the deployment of the software to a staging or production environment.

4) what is clean life cycle? Why it is important in build?

Software development life cycle (SDLC):

The SDLC represents the series of phases or stages that a software project goes through from its initiation to the delivery of the final product. A typical SDLC includes stages such as requirements gathering, design, implementation, testing, deployment, and maintenance. The purpose of following a defined SDLC is to ensure a systematic and structured approach to software development.

There are three built-in build lifecycles:

* default - handles project deployment
* clean - handles project cleaning
* site - handles creation of your project's web site.

In Maven, the "clean" phase is a part of the build lifecycle. when you execute the command 'mvn clean', maven removes the target directory and its contents. The "clean" phase ensure that the project is in a clean state before starting a new build. The target directory typically contains the compiled classes, generated artifacts, and other build-related files. Cleaning the project is useful to eliminate any remnants from previous builds and start with a fresh environment.

Importance of Clean phase:

* Preventing Build Artefact Pollution: Over successive builds, the target directory may accumulate files from previous compilations. Cleaning the project before a new build prevents artifacts from interfering with the current build, ensuring that the build starts with a clean slate.

* Consistency in Build results: Cleaning the project helps maintain consistency in build results. It ensures that the build process is not influenced by any residual files or artifacts left over from previous builds.

* Avoiding side effects: In some cases, changes to the build configuration or dependencies might not take effect unless the project is cleaned. By running 'mvn clean', developers can avoid potential side effects and ensures that the build reflects the latest changes.

* Enforcing build from source: cleaning the project and removing the target directory reinforces the principles of building from source. It discourages relying on previously generated artifacts and promotes a clean, reproducible build process.

5) State any 10 Git command and its usage.

Uses

* git init : Initializes a new Git repository.
* git clone : Creates a copy of a remote repository on your local machine.
* git add : Adds changes in your working directory to the staging area.
* git commit : Records changes from the staging area to the local repository.
* git pull : Fetches changes from a remote repository and merges them into the current branch.
* git push : Pushes local changes to a remote repository.
* git branch : Lists existing branches or creates a new branch.
* git checkout : Switches between branches or restores working tree files.
* git merge : Combines changes from different branches.
* git log : Displays a log of all commits in the current branch.

6) What do you understand about the release management? Explain the release management process with neat diagram.

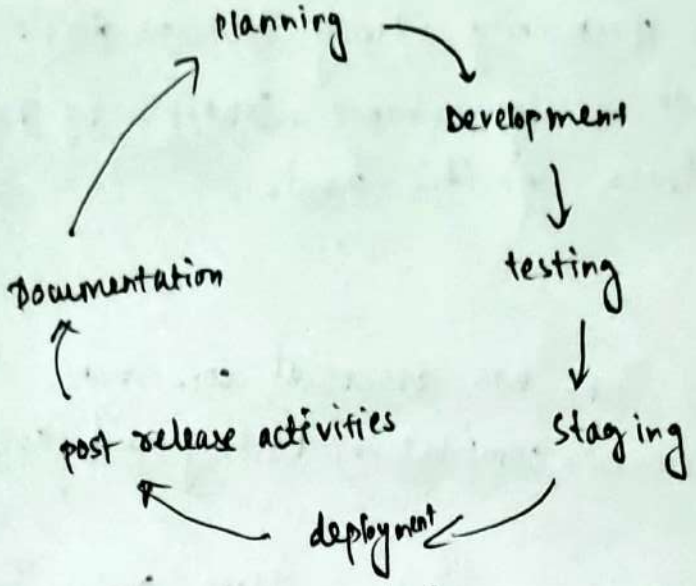
Release management:

Release management is a set of processes and practices aimed at planning, scheduling, and controlling the release of software applications, ensuring that they meet organizational objectives and user expectations. The goal is to deliver high-quality software to end-users in a timely and efficient manner while minimizing risks and disruptions.

Release management process:

The release management process involves several stages, each with specific activities and checkpoints. Below is an overview of the release management process, accompanied by a simplified diagram:

* Planning:
  - Activities
    - → Define release scope and objectives.
    - → Identify features and fixes to be included
    - → Plan resources and timelines.
* Development:
  - → Implement new features and bug fixes.
  - → Conduct unit testing
* Testing:
  - → Perform integration testing
  - → Conduct system testing
  - → Validate against acceptance criteria.
* Staging:
  - → Prepare a staging environment
  - → Perform user acceptance testing (UAT)
  - → Validate the release candidate.
* Deployment:
  - → Deploy the release to production
  - → Monitor for any issues or disruptions.
* Post-release activities:
  - → Conduct post-implementation review
  - → Collect user feedback.
  - → Address any issues or bugs promptly
* Documentation:
  - → Update release notes and documentation.
  - → Capture lessons learned for future releases.



*) Briefly discuss Jenkins framework and its usage.

Jenkins is an open-source automation server that facilitates building, testing and deploying code by automating parts of the software development process. It is widely used for continuous integration (CI) and continuous delivery (CD) purposes, enabling developers to automate repetitive tasks and streamline the software development lifecycle.

Key features of Jenkins:

* Automation: Jenkins automates the building, testing and deployment of software projects. It allows developers to define pipelines, which are sequences of automated actions to be executed.

* Integration: Jenkins integrates with a wide range of plugins, tools and technologies. This extensibility makes it suitable for diverse development environments and stacks.

* Continuous Integration/Continuous Deployment (CI/CD): Jenkins supports the CI/CD paradigm by automating the integration of code changes, running tests, and deploying applications. This results in faster and more reliable software delivery.

* Scalability: Jenkins is scalable and can be used to manage and coordinate complex build and deployment scenarios across

multiple machines and environments.

*Extensibility: The Jenkins community actively develops plugins that extend its functionality. Users can easily enhance Jenkins by installing plugins that cater to their specific needs.

Usage of Jenkins:

*Setting up Jobs: Jenkins jobs are essential components defining steps, including code retrieval, compilation, testing and application deployment.

*Building code: Jenkins automates code building, pulling source code, compiling it, and generating executable artifacts.

*Running test: Jenkins executes automated tests in CI/CD pipelines to ensure code changes meet quality standards without introducing regressions

*Continuous deployment: Jenkins seamlessly integrates with version control systems like Git, SVN, triggering builds based on code changes.

*Notification and Reporting: Jenkins notifies team members of build and deployment result via email or messaging platforms and generates reports for process insights.

*Parameterized Builds: Jenkins supports parameterized builds, allowing users to customize job configurations by passing parameters dynamically.

*Pipeline as code: Jenkins facilitates defining pipelines as code using DSL or tools like Jenkinsfile, promoting version control and sharing of build and deployment pipelines.

6) Briefly discuss Maven Build tool and its usage.

Maven is a popular open-source build and project management tool primarily used for JAVA projects, but it can be adapted for projects in other languages. It simplifies the process of building and managing projects by providing a standard structure, project dependencies management, and build lifecycle.

Key features of Maven :

* Project Object Model (POM) : Maven uses a project object model, defined in an XML file (pom.xml), to manage project configuration, dependencies and build settings. The POM serves as a blueprint for the project.

* Dependency management : Maven simplifies dependency management by automatically downloading and managing project dependencies from a central repository. This eliminates the need for developers to manually download and configure libraries.

* Build lifecycle : Maven defines a set of build phases and goals that constitute its build lifecycle. common phase include clean, compile, test, package, install, deploy. Developers can execute specific phases to perform tasks at different stages of the build process.

* Plugin : Maven uses plugins to extend its functionality. Plugins are configured in the POM file and can be used to execute specific tasks during the build process, such as compiling code, running tests, generating documentation, and more.

* Consistent project structure : Maven enforces a standardized project structure, making it easier for developers to understand and contribute to projects following the Maven convention. This structure promotes consistency across different projects.

* Central repository: Maven relies on a central repository to store and distribute libraries and plugins. This repository, known as the Maven Central Repository, is a vast collection of pre-built libraries that developers can include in their projects.

Uses:

* Continuous Integration: Automating builds whenever changes are pushed to version control.
* Project initialization: Developers can use the 'mvn archetype:generate' command to initialize a new Maven project by selecting a project template.
* Dependency management: Developers specify project dependencies in the POM file, and Maven automatically downloads and manages the required libraries from the central repository.
* Building projects: Developers utilize the 'mvn clean install' command to compile source code, run tests, package the application, resulting in JAR or WAR files.
* Running tests: Maven supports unit test execution with the 'mvn test' command, generating test reports available in the target directory.
* Generating documentation: Maven can generate project documentation, such as Javadocs, through the 'mvn javadoc:javadoc' command.
* Creating a distribution: Maven aids in creating distribution packages (ZIP or TAR files) with compiled code and dependencies using the 'mvn package' command.
* Integration with IDEs: Maven seamlessly integrates with IDEs like Eclipse, IntelliJ IDEA, and NETBeans, enabling developers to import and work with Maven projects effortlessly.

a) A computer-aided design application for mechanical components with following 7 functions in developed. Compute the metrics - project cost & effort involved.

Function	optimistic	most likely	pessimistic
1	5000	1600	4800
2	6700	5000	5100
3	4600	6900	8600
4	4200	3000	3900
5	7700	4000	6000
6	4600	1000	4000
7	9000	8000	9400

ED = (Optimistic + 4 x Most Likely + Pessimistic)/6

~~(Estimation~~
(Expected duration)

ED function

1 = 5000 + 4x1000 + 4800/6 = 2300

2 = 6700 + 4x5000 + 5100/6 = 5300

3 = 4600 + 4 x 6900 + 8600/6 = 6600

4 = 4200 + 4 x 3000 + 3900/6 = 3350

5 = 7700 + 4 x 4000 + 6000/6 = 4950

6 = 4600 + 4 x 1000 + 4000/6 = 2100

7 = 9000 + 4 x 8000 + 9400/6 = 8400

Total Expected duration (TED)

$$= \sum ED_{function}$$

$$= \frac{2300 + 5300 + 6800 + 3350 + 4950 + 2400 + 8400}{7}$$

$$= 4742.85$$

Project cost (PC) = C x TED

C → cost per unit duration

Project effort (PE) = $\frac{TED}{C}$

Let cost be (C) - ₹ 50

PC = C x TED

PE = $\frac{TED}{C}$

then

PC = $50 \times 4742.85 = 237142.5$

PE = $\frac{4742.85}{50} = 94.857$ p-m

10) A project was estimated with size of 300 KLOC. Calculate effort, scheduled time development. Also, calculate the Average resource size and productivity of the software for (3) project types.

Project type	a	b	c	d
organic	2.4	1.05	2.5	0.38
semidetache	3	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

(Effort) $E = a \times (KLOC)^b$

(Scheduled time) $T = c \times (E)^d$

(Avg. resource size) $RS = E/T$

(Productivity) $P = KLOC/E$

Organic project:

$E_{organic} = 2.4 \times (300)^{1.05} = 957.609$

$T_{organic} = 2.5 \times [E_{organic}]^{0.38} = 33.946$
(957.609)

$RS_{organic} = E_{organic} / T_{organic} = \frac{957.609}{33.946} = 28.209$

$P_{organic} = 300 / E_{organic} = 0.313$

Semidetached project:

$E_{semidetached} = 3 \times (300)^{1.12} = 1784.41$

$T_{semidetached} = 2.5 \times (1784.41)^{0.35} = 34.352$

$RS_{semidetached} = 1784.41 / 34.352 = 51.944$

$P_{semidetached} = 300 / 1784.41 = 0.168$

Embedded project :

$E_{embedded} = 3.6 \times (300)^{1.2} = 3379.46$

$T_{embedded} = 2.5 \times (3379.46)^{0.32} = 33.664$

$RS_{embedded} = 3379.46 / 33.664 = 100.387$

$P_{embedded} = 300 / 3379.46 = 0.0887$