

Implementing Software Methodology (SCQF level 8)_J6V348

Guidance Notes:

Before we start - Let's discuss fair use of AI software

Tip! – I used AI to generate the guideline for the first question – We have no problem with you using a similar tool as a research technique but you must then extrapolate and combine this with other research so that your answer reflects your own understanding and is refactored into your own wording.

Obviously, It can be an extremely useful tool for grasping and enhancing your understanding of theory (and will be a valid tool to utilise in the workplace – It would therefore be ridiculous to ignore its potential).

Similarly it can be used to perhaps give you an example of a code snippet if you are struggling with a particularly tricky concept but should not be used to generate entire passages of your code. ie: you might ask it to "show me an example of how to establish a JDBC connection to a MySQL database. This will give you a guideline which you can interpret and adapt to make your own connection to your own database.

So, **“DO NOT”** try to pass off AI generated text or code samples as your own work this will not reflect your own understanding, is actually quite easy to detect and could result in you losing your apprenticeship.

Here is what an alternative AI software thought of my answer –



Yes, the answer provided appears to have been generated by AI software!

YOU MAY FREELY QUOTE FROM AI GENERATED ANSWERS and discuss the relevance of the information you have researched

YOU MUST, HOWEVER, REFERENCE THE AI SITE AND DATE OF USE.

Performance requirements

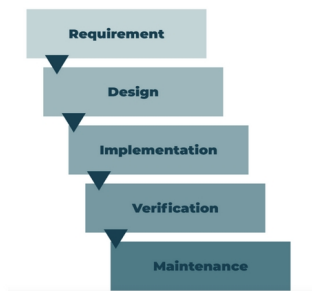
1. Implementing appropriate software development methodologies including predictive (plan-driven) approaches or adaptive (iterative / agile) approaches.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Implementing appropriate software development methodologies involves choosing between predictive (plan-driven) approaches and adaptive (iterative/agile) approaches.

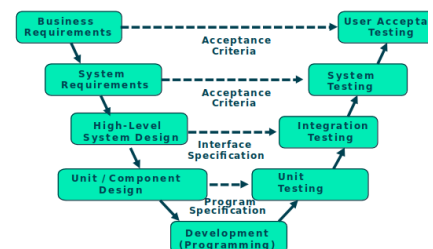
- **Predictive (Plan-Driven) Approaches:**
 - Involves detailed planning and documentation before starting development.
 - Emphasis is on following a strict plan and timeline.
 - Suitable for projects with well-defined requirements and stable scope.
 - Examples include Waterfall and V-Model methodologies.

Waterfall



V-Model

- V-Model of development develops a test at each stage
- Requirements gathering and analysis System testing
- High-level design of modules Integration tests
- Low-level code design Unit tests
 - Ensures no requirement is missed or ignored
 - Can be used in iterative phases with each phase following a V-model testing strategy

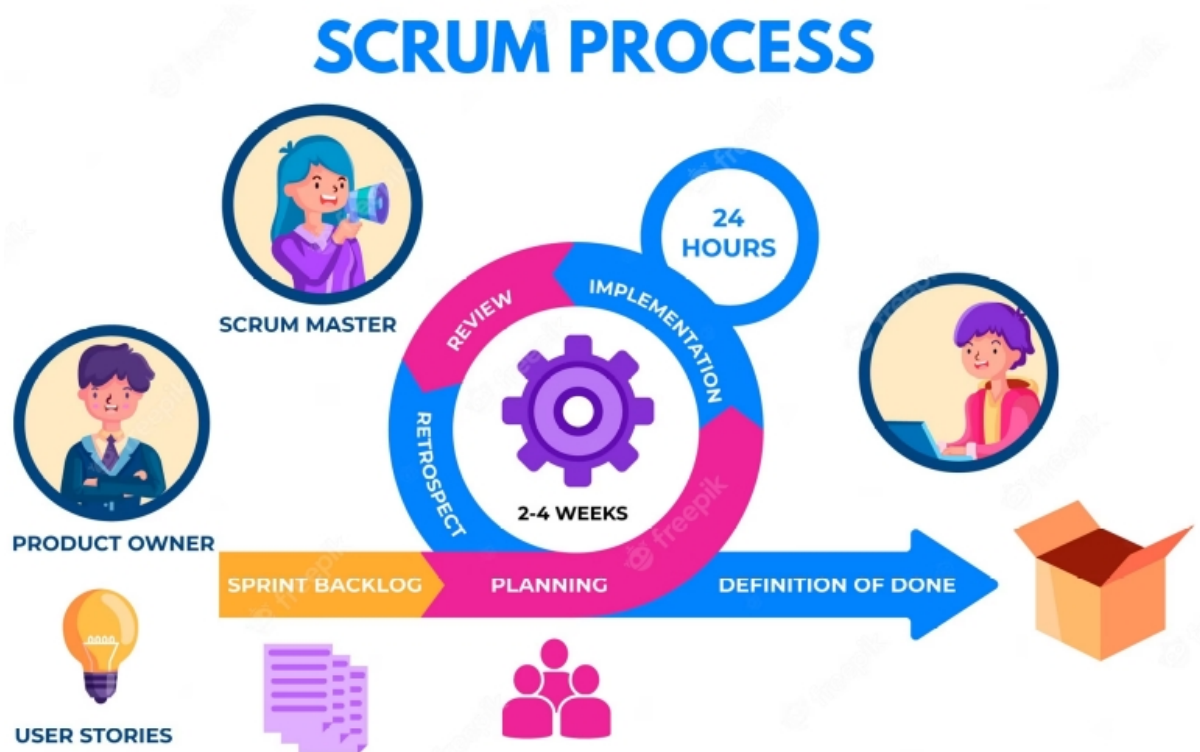


The Waterfall & V-Model methodologies - Each phase must be wholly completed before progressing. In the case of the V-Model testing must be completed at each step.

- **Adaptive (Iterative/Agile) Approaches:**

- Focuses on flexibility and adapting to changes during development.
- Works in short iterations or sprints, allowing for continuous feedback and improvement.
- Ideal for projects with evolving requirements and uncertain scope.
- Examples include Scrum, Kanban, and Extreme Programming (XP) methodologies.

By understanding the differences between these methodologies, software development teams can choose the approach that best fits the project's needs, ensuring efficient and successful development processes.



Source: Image by Freepik

Here is an image describing the Scrum Process – This is considered an Agile Approach to software development where changes are welcomed and short “Sprints” or iterations lead to releasable beneficial versions rather than taking a full journey approach

2. Identifying user requirements and translating these into user stories.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Identifying user requirements and translating them into user stories is a crucial step in software development. Here are some best practices to achieve this effectively:

- **Engage with Stakeholders:**
 - Collaborate with stakeholders, including end-users, customers, and product owners, to gather requirements.
 - Conduct interviews, surveys, workshops, and focus groups to understand their needs and expectations.
- **Use Different Techniques:**
 - Employ various techniques like brainstorming, user interviews, prototyping, and surveys to uncover requirements.
 - Use tools like personas, user journey maps, and use case diagrams to visualize and understand user needs.
- **Document Requirements:**
 - Document requirements in a clear, concise, and structured manner.
 - Use tools like requirement documents, user requirement specifications, and functional specifications to capture and organize requirements.
- **Prioritize Requirements:**
 - Prioritize requirements based on their importance, impact, and feasibility.
 - Use techniques like MoSCoW prioritization (Must have, Should have, Could have, Won't have) to categorize requirements.
- **Translate into User Stories:**
 - Write user stories that are simple, concise, and focused on user needs.
 - Follow the INVEST criteria (Independent, Negotiable, Valuable, Estimable, Small, Testable) when writing user stories.
- **Include Acceptance Criteria:**
 - Define acceptance criteria for each user story to clarify what needs to be done to consider the story complete.
 - Use acceptance criteria to ensure all aspects of the user story are understood and met.

By following these best practices, you can effectively identify user requirements, capture them in user stories, and ensure that the development team understands and delivers what the users truly need.



Here you can see an example of a user Journey - Harry wishes to register as a student for an on-line training application.

3. Working as part of a software development team to develop and deliver software solutions.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

When working as part of a software development team to develop and deliver software solutions, you may take on various roles depending on your expertise and the project requirements. Some common roles in a software development team include:

1. **Project Manager:**

- Responsible for overall project planning, execution, and delivery.
- Manages resources, timelines, and budgets.
- Ensures project goals align with business objectives.

2. **Product Owner:**

- Represents the stakeholders and customers.
- Defines and prioritizes features based on user needs.
- Works closely with the development team to ensure the product meets requirements.

3. **Software Developer/Engineer:**

- Designs, codes, tests, and maintains software solutions.
- Collaborates with team members to implement features.
- Ensures code quality, scalability, and performance.

4. **Quality Assurance (QA) Engineer:**

- Tests software to identify bugs, defects, and issues.
- Develops test plans and executes test cases.
- Works to ensure the software meets quality standards.

5. **UI/UX Designer:**

- Designs user interfaces and user experiences.
- Creates wireframes, mockups, and prototypes.
- Focuses on usability, accessibility, and user satisfaction.

6. **DevOps Engineer:**

- Manages the development infrastructure and deployment processes.
- Automates build, test, and deployment tasks.
- Ensures continuous integration and continuous delivery practices.

7. Scrum Master:

- Facilitates Agile practices within the team.
- Helps the team follow Scrum processes and ceremonies.
- Removes obstacles and fosters collaboration.

8. Technical Architect:

- Designs the overall structure of the software solution.
- Defines technology stack, components, and integrations.
- Ensures the system's scalability, security, and performance.

By taking on these roles within a software development team, each member contributes to the successful development and delivery of software solutions by leveraging their expertise and collaborating effectively with teammates.

4. Meeting with stakeholders/project owners to agree on deliverables.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

When meeting with stakeholders or project owners to agree on deliverables, it's essential to follow a structured process to ensure clarity, alignment, and mutual understanding. Here is a typical process for this meeting:

- **Preparation:**

Clarify the meeting objectives and desired outcomes.
Review the project scope, requirements, and any relevant documentation.
Prepare a meeting agenda outlining topics to be discussed.

- **Introduction:**

Introduce all meeting participants and their roles.
Set the context by providing a brief overview of the project goals and objectives.
Establish a positive and collaborative atmosphere.

- **Review of Requirements:**

Present the project requirements, goals, and constraints.
Discuss any changes or updates to the initial requirements.
Seek clarification on any ambiguous or unclear points.

- **Proposed Deliverables:**

Present the proposed deliverables, including features, functionality, and timelines.
Explain how each deliverable aligns with the project objectives.
Discuss any dependencies or risks associated with the deliverables.

- **Feedback and Discussion:**

Encourage stakeholders to provide feedback on the proposed deliverables. Address any concerns, questions, or suggestions raised by the stakeholders.

Discuss trade-offs, priorities, and potential changes based on feedback.

- **Alignment and Agreement:**

Work towards consensus on the final set of deliverables.

Ensure that all stakeholders agree on the scope, timeline, and quality expectations.

Document the agreed-upon deliverables in a formal agreement or project plan.

- **Next Steps:**

Summarize the key decisions and action items from the meeting.

Define next steps, responsibilities, and timelines for moving forward.

Confirm any follow-up meetings or communication channels.

- **Closure:**

Thank the stakeholders for their time, input, and collaboration.

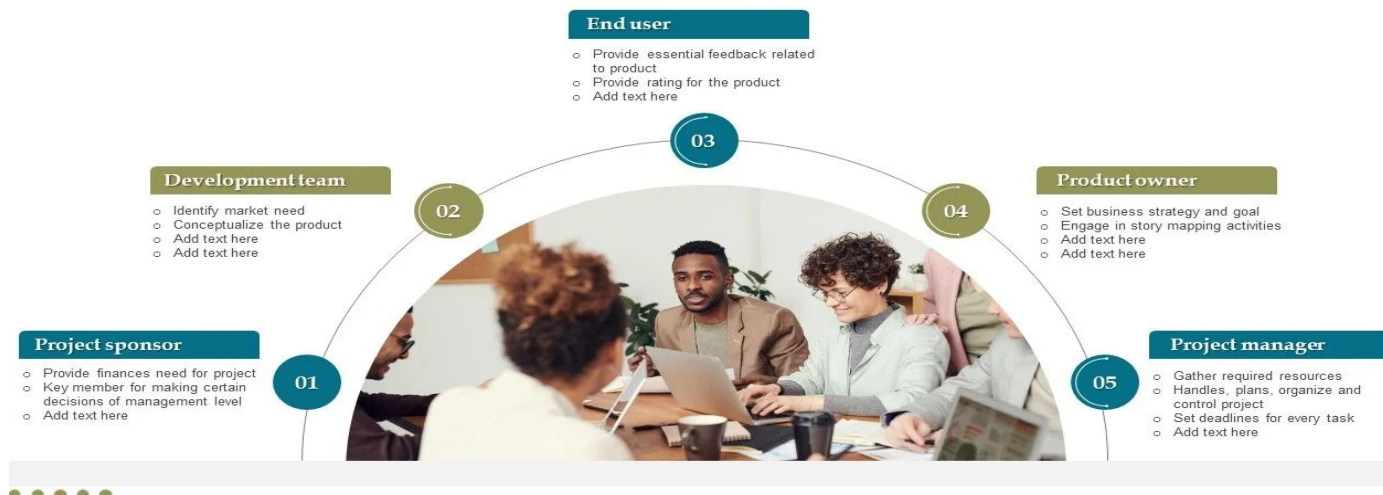
Confirm understanding of the agreed deliverables and expectations.

Provide contact information for further questions or clarifications.

By following this process, you can effectively engage with stakeholders and project owners to reach a consensus on project deliverables, ensuring that everyone is aligned and committed to achieving the project's goals.

Stakeholders roles in agile service delivery

The following slide showcases key roles of stakeholders in agile product delivery to assist organization with smooth operations. Main stakeholders included are project sponsor, product owner, development team, manager, business analyst and end user.



Main stakeholders included are project sponsor, product owner, development team, manager, business analyst and end user. <https://www.slidegeeks.com/stakeholders-roles-in-agile-service-delivery-ppt-icon-example-topics-pdf> - As at 6/8/24

5. **Employing version control of source code and related artefacts to manage the development and configuration of source code, tested software, and documents.**

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Employing version control of source code and related artefacts is crucial for managing the development and configuration of source code, tested software, and documents efficiently. Here is an overview of the process involved in utilizing version control:

1. Choose a Version Control System (VCS):

- Select a suitable version control system such as Git, Subversion (SVN), or Mercurial based on your team's needs and preferences.
- Consider factors like distributed vs. centralized systems, branching and merging capabilities, and community support.

2. Setup the Version Control System:

- Create a new repository for your project in the chosen VCS.
- Configure access control and permissions for team members based on their roles.
- Initialize the repository with the initial codebase, if applicable.

3. Track Changes:

- Add source code files, documentation, configuration files, and other artefacts to the version control system.
- Commit changes to the repository with descriptive messages explaining the purpose of each change.
- Regularly update the repository with new code additions, modifications, and deletions.

4. Branching and Merging:

- Create branches for different features, bug fixes, or experiments to isolate changes.
- Merge branches back into the main branch (e.g., master or main) after code review and testing.
- Resolve conflicts that may arise during the merging process.

5. Collaboration:

- Collaborate with team members by sharing code through the version control system.

- Review and comment on each other's code using pull requests or merge requests.
- Discuss changes and provide feedback within the VCS platform.

6. Code Review:

- Conduct code reviews to ensure code quality, adherence to coding standards, and correctness.
- Use tools integrated with the version control system for code review workflows.
- Address feedback and make necessary revisions based on code review comments.

7. Automate Testing:

- Integrate automated testing processes into the version control workflow.
- Trigger test runs on each code commit to identify issues early.
- Use continuous integration (CI) tools to automate testing and deployment tasks.

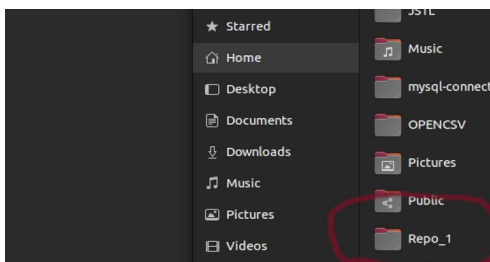
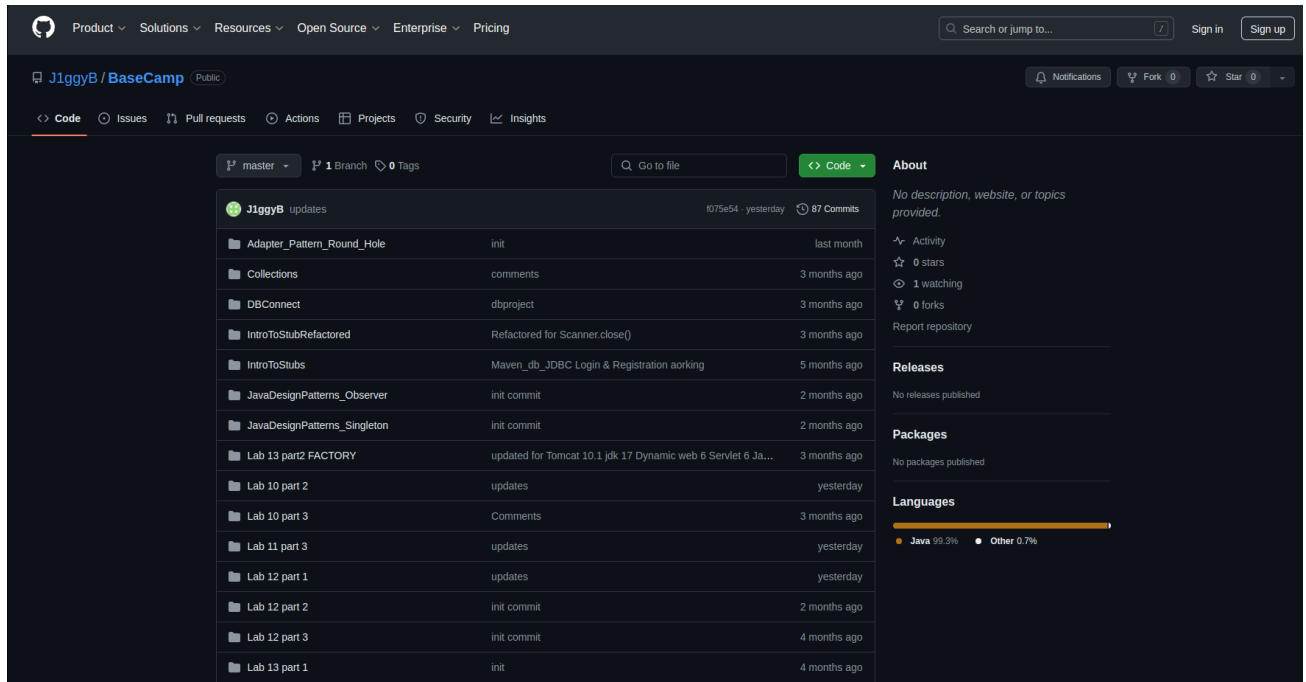
8. Documentation and Artefacts:

- Version control not only source code but also documentation, configuration files, and other project artefacts.
- Keep documentation up-to-date and in sync with code changes.
- Ensure that all project-related files are managed and versioned properly.

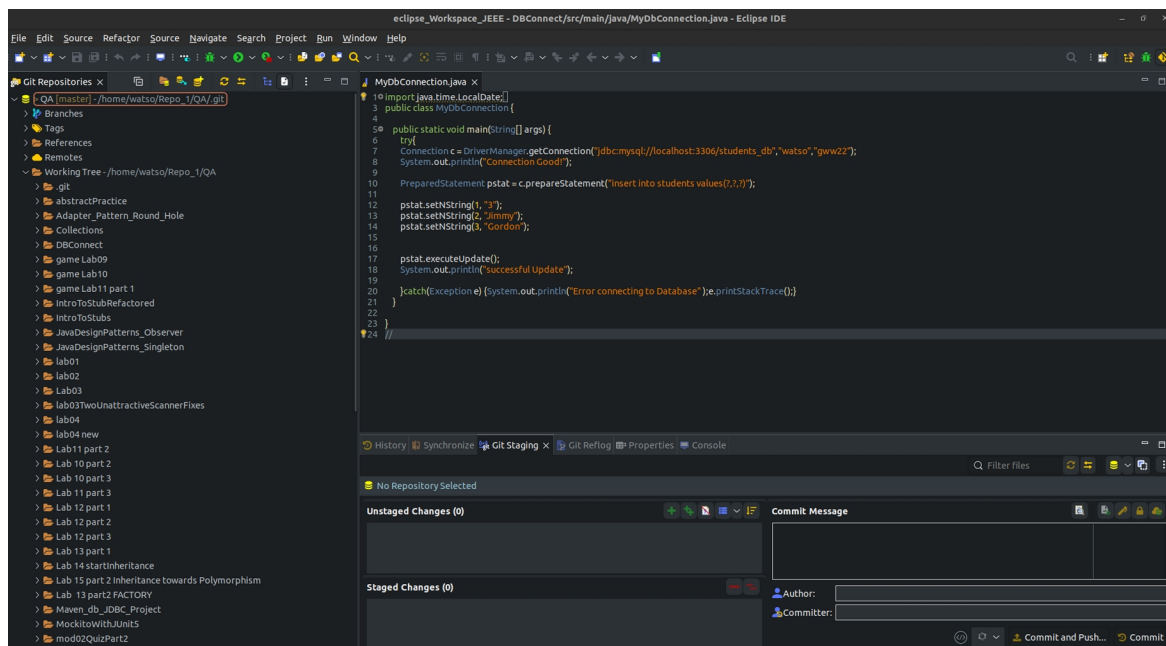
9. Release Management:

- Tag specific commits or branches to mark releases or milestones.
- Create release notes detailing changes included in each release.
- Deploy releases to production environments from version-controlled codebases.

By following these steps and best practices, teams can effectively utilize version control systems to manage the development and configuration of source code, tested software, and documents, leading to improved collaboration, traceability, and project scalability.



Here you can see I have created a Git Hub Account associated with a repository folder on my PC.



Here you can see that I have linked my Eclipse IDE to my Git Repository

6. Working within a Continuous Integration (CI) capability.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Continuous Integration (CI) is a software development practice where developers frequently integrate code changes into a shared repository. This approach automates the process of building and testing code, enabling teams to detect and address integration errors early. Here are the benefits and steps involved in working within a Continuous Integration capability:

Benefits of Continuous Integration (CI):

- **Early Detection of Issues:** CI helps identify integration errors, bugs, and conflicts as soon as code is committed, allowing teams to address them promptly.
- **Faster Feedback Loop:** Developers receive immediate feedback on code changes, enabling them to iterate quickly and improve code quality.
- **Improved Code Quality:** Regular integration and automated testing lead to higher code quality, reducing the likelihood of bugs in production.
- **Streamlined Development Process:** CI automates build, test, and deployment processes, making development more efficient and reliable.
- **Enhanced Collaboration:** CI encourages collaboration among team members by promoting frequent code integration and communication.
- **Risk Mitigation:** By catching issues early, CI reduces the risk of introducing defects and regressions into the codebase.
- **Scalability:** CI facilitates scaling development efforts by providing a consistent and reliable process for integrating code changes.

Steps Involved in Working with Continuous Integration (CI):

1. Setup CI Environment:

- Choose a CI tool such as Jenkins, Travis CI, CircleCI, or GitLab CI/CD.
- Configure the CI tool to connect to your version control system and trigger builds on code changes.

2. Automate Build Process:

- Create build scripts (e.g., using Maven, Gradle) to compile code, run tests, and package the application.
- Configure the CI tool to execute these build scripts automatically on each code commit.

3. Automate Testing:

- Write automated tests (unit tests, integration tests, etc.) to validate the functionality and quality of the application.
- Integrate automated tests into the CI pipeline to run them automatically during the build process.

4. Continuous Integration Pipeline:

- Define a CI pipeline that includes stages for building, testing, and deploying the application. (Software such as Jenkins)
- Configure the pipeline to run sequentially, with each stage triggering the next upon successful completion.

5. Code Quality Checks:

- Include static code analysis tools (e.g., SonarQube) in the CI pipeline to check code quality, identify code smells, and enforce coding standards.

6. Feedback and Notifications:

- Set up notifications to alert developers of build failures, test errors, or other issues.
- Ensure that feedback is provided promptly so that developers can address issues quickly.

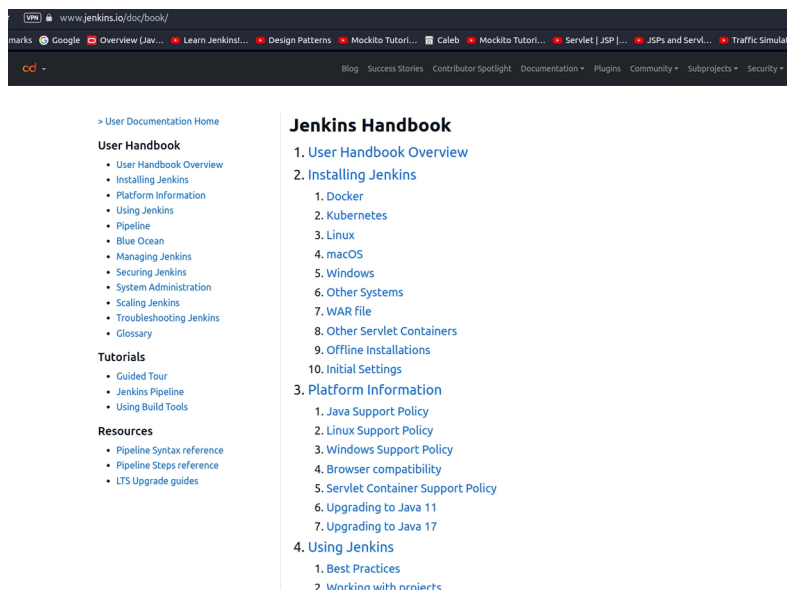
7. Version Control Integration:

- Ensure that the CI tool is integrated with the version control system to trigger builds on code changes.
- Use feature branches and pull requests to isolate changes and facilitate code reviews before merging.

8. Deployment Automation:

- Automate deployment to development, staging, and production environments as part of the CI pipeline.
- Ensure that deployments are consistent, repeatable, and traceable.

By following these steps and leveraging the benefits of Continuous Integration, development teams can improve code quality, collaboration, and efficiency while mitigating risks associated with software development.



Here you can see the “Documentation” tab of the Jenkins website which explains how to set up a CI Pipeline. <https://www.jenkins.io/doc/book/> - As at 6/8/24

7. **Agreeing development tasks, responsibilities, and priorities with the development team.**

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

When agreeing on development tasks, responsibilities, and priorities with the development team, it is essential to establish clear communication and alignment to ensure successful project execution. Here is a structured process to facilitate this agreement:

Process of Agreeing Development Tasks, Responsibilities, and Priorities:

1. Initial Planning Meeting:

- **Schedule a Meeting:** Arrange a meeting with the development team to discuss upcoming tasks and priorities.
- **Set Objectives:** Define the purpose of the meeting, outlining the need to agree on development tasks, responsibilities, and priorities.

2. Task Identification:

- **List Tasks:** Compile a list of development tasks based on project requirements, user stories, and feedback.
- **Break Down Tasks:** Divide tasks into smaller, manageable units to facilitate prioritization and allocation.

3. Responsibility Assignment:

- **Define Roles:** Clarify team members' roles and responsibilities within the development process.
- **Assign Tasks:** Match each task with team members based on their skills, expertise, and availability.

4. Prioritization:

- **Establish Criteria:** Define criteria for task prioritization, such as impact on project goals, dependencies, deadlines, and risks.
- **Rank Tasks:** Prioritize tasks collaboratively with the team, considering input from developers, project managers, and stakeholders.

5. Task Agreement:

- **Discuss Tasks:** Review each task, its requirements, and expected outcomes with the team.
- **Confirm Understanding:** Ensure that team members understand the tasks, dependencies, and timelines associated with their assignments.

6. Responsibility Clarity:

- **Set Expectations:** Clearly communicate expectations regarding task completion, quality standards, and collaboration.
- **Address Questions:** Encourage team members to ask questions and seek clarification on tasks and responsibilities.

7. Alignment on Priorities:

- **Consensus Building:** Reach a consensus on task priorities by discussing trade-offs, resource constraints, and project goals.
- **Document Decisions:** Record agreed-upon priorities, responsibilities, and timelines to maintain alignment.

8. Feedback Mechanism:

- **Feedback Loop:** Establish a feedback mechanism for team members to provide updates on task progress, raise concerns, and seek assistance.
- **Regular Check-ins:** Schedule periodic check-ins to review progress, adjust priorities if needed, and address any issues that may arise.

9. Documentation:

- **Task Documentation:** Document agreed-upon development tasks, responsibilities, and priorities in a shared project management tool or document.
- **Accessibility:** Ensure that task details, responsibilities, and priorities are accessible to all team members for reference.

By following this structured process and fostering open communication and collaboration within the development team, you can effectively agree on development tasks, responsibilities, and priorities, setting the stage for successful project delivery.

[illegible]

One example of agreeing development tasks, responsibilities, and priorities with the development team can be found in Scrum where the development team produce a “Sprint Backlog”. The result of a Sprint Planning meeting is a Sprint Backlog that details the work to be carried out in the Sprint. This includes a plan of how it will be Done and a Sprint Goal that explains why the work is valuable. Above you can see a template for a “Sprint Backlog”.

https://www.stakeholdermap.com/agile-templates/sprint-backlog-template.html#google_vignette – As at 6/8/24

8. Operating within a sprint or development phase to deliver within timescales.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Operating within a sprint or development phase to deliver within timescales involves following a structured approach to ensure efficient and timely delivery of software solutions. Here is the process explained:

1. **Sprint Planning:**

- **Define Goals:** Start by setting clear goals and objectives for the sprint or development phase.
- **Prioritize Tasks:** Identify and prioritize tasks based on their importance and impact on the project timeline.

2. **Daily Stand-ups:**

- **Communicate Progress:** Hold daily stand-up meetings to discuss progress, challenges, and updates on tasks.
- **Address Roadblocks:** Identify and address any obstacles that may hinder progress.

3. **Collaborative Development:**

- **Team Collaboration:** Foster collaboration among team members to ensure smooth coordination and alignment.
- **Feedback Loop:** Encourage feedback and communication to iterate on solutions and make necessary adjustments.

4. **Continuous Integration:**

- **Automated Testing:** Implement automated testing to ensure code quality and identify issues early.
- **Regular Integration:** Continuously integrate code changes to detect and resolve conflicts promptly.

5. **Iterative Development:**

- **Feedback Incorporation:** Incorporate feedback from stakeholders and users to refine and improve the product.
- **Iterate and Improve:** Iterate on features based on feedback and testing results to enhance the product.

6. **Time Management:**

- **Time-boxing:** Allocate specific time-frames for tasks and activities to manage time effectively.
- **Monitor Progress:** Regularly track progress against timelines to ensure timely delivery.

By following these steps and principles of agility, collaboration, innovation, and user-centricity, teams can operate within sprints or development phases to deliver software solutions within timescales effectively.

APMG International

OUR SERVICES - ABOUT US PARTNERS CONTACT - Q

Top The article Related products Reading All news

Benefits of Using Scrum

Below we explore the top 10 advantages of using Scrum in your projects. From its inherent flexibility to its focus on continuous improvement, Scrum offers a plethora of benefits that can transform the way your team works and delivers products.

- 1. Flexibility:**
 Scrum's iterative and incremental approach allows teams to adapt to changing requirements and priorities throughout development.

 Every 1-4 weeks, depending on the duration of your Sprint, the Scrum Team decide what will be worked on in the next Sprint. At any time prior to planning the work of a Sprint, the Product Owner collaborates with a number of Developers to refine the Product Backlog, ensuring that the most valuable work is available for selection, whether that is pre-existing in the backlog or newly added.
- 2. Faster Time to Market:**
 By delivering 'Done' Increments of the Product in short sprints, Scrum allows value to be realised sooner, reducing time-to-market and allowing for quicker feedback from customers. An Increment is born as soon as it meets its Definition of Done – this can be at any time during the Sprint or at the end. If the Definition of Done describes a releasable Increment of the Product, then it can start realising value as soon as it is released.
- 3. Enhanced Collaboration:**
 Scrum promotes a collaborative environment through the application of the Scrum Values.

 The Scrum events, the artefacts with their commitments, and clear accountabilities associated with the Scrum roles help bring this to life. Developers in a Scrum Team are multifunctional, meaning that most of the work can be done by most people in the team. For more complicated aspects of the work, e.g. designing solutions and solving problems, a collaboration between two or three Developers will lead to better outcomes, often achieved quicker than one person working alone.

The above web-site discusses 10 benefits of taking an Agile approach to software development by adopting the SCRUM principles and practices.
<https://apmg-international.com/article/top-10-benefits-scrum-and-frequent-challenges> -As at 6/8/24

9. Agreeing software completion end and sign-off arrangements.

Sample Answer: - For reference only - Lists items you could include in your discussion paragraph

Agreeing on software completion, end, and sign-off arrangements is a critical step in the software development process to ensure that all stakeholders are aligned on the delivery of the final product. Here is a detailed process for this agreement:

Process of Agreeing Software Completion, End, and Sign-off Arrangements:

1. Final Product Review:

- **Internal Review:** Conduct a comprehensive review of the software to ensure it meets all requirements and specifications.
- **User Acceptance Testing (UAT):** Involve stakeholders in UAT to validate that the software functions as expected and meets user needs.

2. Bug Fixing and Refinement:

- **Address Issues:** Address any bugs, defects, or issues identified during testing and reviews.
- **Refinement:** Make necessary refinements to enhance the software's performance, usability, and overall quality.

3. Documentation Verification:

- **Documentation Review:** Ensure that all necessary documentation, including user manuals, technical specifications, and release notes, is accurate and up to date.
- **Legal Compliance:** Verify compliance with any legal or regulatory requirements related to the software.

4. Performance Evaluation:

- **Performance Testing:** Conduct performance testing to assess the software's speed, stability, and scalability.
- **Security Assessment:** Validate that the software is secure and free from vulnerabilities that could compromise data or functionality.

5. End-User Training:

- **Training Sessions:** Provide end-user training sessions to familiarize users with the software's features, functionalities, and best practices.
- **Support Documentation:** Deliver support documentation to help users troubleshoot common issues and maximize software utilization.

6. Sign-off Process:

- **Agreement on Completion:** Schedule a meeting with stakeholders to review the software's readiness for deployment.
- **Formal Sign-off:** Obtain formal sign-off from key stakeholders, indicating their acceptance of the software's completion and readiness for release.




7. Transition Plan:

- **Deployment Strategy:** Develop a deployment plan outlining how the software will be rolled out to end users.
- **Transition Support:** Provide transition support to assist users in adopting the new software and addressing any initial challenges.

8. Post-Implementation Review:

- **Feedback Collection:** Gather feedback from users post-implementation to assess the software's performance and user satisfaction.
- **Iterative Improvements:** Use feedback to identify areas for improvement and plan for future iterations or updates.

By following this process, software development teams can ensure a smooth transition from development to deployment, secure stakeholder buy-in, and achieve successful software completion, end, and sign-off arrangements.

HELPING SMALL BUSINESSES SUCCEED				MORE DONUTS:	START UP	MARKETING	MONEY	TECH
LAW Donut		BLOG	NEWS	OFFERS	FOLLOW US   			
Starting up	Sector specific law	Business ownership and management	Finance and business strategy	Employment law	Marketing and selling	MORE ...		

Legal issues in software development FAQs

Which are the legal issues involved in software development?

Legal issues in software development are diverse and complex, primarily focusing on:

- **Intellectual property rights:** Protecting creations with copyrights, patents, and trademarks, and ensuring non-infringement of others' rights.
- **Licensing agreements:** Comprehending and adhering to software licensing terms, including both open source and proprietary licenses.
- **Data privacy and protection:** Complying with data protection regulations like GDPR and CCPA and ensuring robust data security.
- **Third-party software and dependencies:** Managing legal compliance and security vulnerabilities associated with third-party components.
- **Software development agreements:** Drafting clear agreements detailing work scope, IP ownership, timelines, and payment terms.
- **Confidentiality agreements:** Protecting sensitive information through binding non-disclosure agreements.
- **Compliance with industry standards:** Adhering to specific regulations relevant to the software's application area, such as HIPAA for healthcare software.
- **Accessibility and anti-discrimination:** Ensuring software is accessible to all users and free from discriminatory biases.
- **Export controls and international laws:** Navigating the complexities of international legal compliance and export controls.
- **Employee and contractor agreements:** Defining clear terms regarding IP rights, confidentiality, and non-compete clauses with staff.

Legal Compliance is a particular concern in software development. The above website hosts an article on the top ten legal issues in software development. <https://www.lawdonut.co.uk/business/blog/24/01/ten-legal-issues-software-development> – As at 6/8/24

Knowledge and understanding

1. Modern techniques used in the design and development of software systems.

(Answered by P1 -9)

2. The different approaches to organising software development using plan driven or adaptive iterative / agile methodologies.

(Answered by P1)

3. How teams work effectively to develop software solutions embracing agile and other development approaches.

(Answered by P1 - 4)

4. The importance of version control and how to apply industry standard solutions.

(Answered by P5)

5. How to implement software development using Continuous Integration.

(Answered by P6)

6. The importance of continual improvement within a dynamic information driven environment.

(Answered by P6 -8)

7. The benefits of frequent merging of code, ensuring all tests pass.

(Answered by P6 & 8)

8. The implementation of automation within software development.

(Answered by P6 & 8)

9. The variety of documents that are required to support software development.

(Answered by P4, 5, 7 & 9)

10. How to provide task estimates to inform software project planning.

(Answered by P2 & 4)

11. How to implement collaborative software development, team-working and organising software development.

(Answered by P1-4 & 7-9)