Work Process

Document Version 2.0

WebCodeGenie Technology Pvt. Ltd.

Table of Content

Scrum in Agile Methodology

Agile methodology is a type of project management process, primarily used for software development, where demands and solutions evolve through the collaborative effort of self-organizing and cross-functional teams. Within the Agile methodology, there are multiple frameworks that can be utilized, one of which is Scrum.

Scrum is a way of working under the Agile method that's mostly used in software development. It's a system that helps teams handle big tasks by breaking them into smaller parts. This way, teams can adjust and solve problems as they come up.

In Scrum, big tasks are broken down into smaller parts called 'User Stories'. These are then worked on in short cycles called 'Sprints', usually lasting 2-4 weeks.

A Scrum team has three roles: a Product Owner, a Scrum Master, and the Development Team. The Product Owner decides what needs to be done and the order of doing it, based on the value to the business. The Scrum Master ensures the team follows Scrum rules and helps remove any roadblocks. The Development Team are the people who do the work to create the product.

Scrum includes regular meetings like Sprint Planning, Daily Stand-ups, Sprint Review, and Sprint Retrospectives. These meetings encourage regular talking, feedback, and continuous improvement. In simple terms, Scrum is an easy but very effective way for teams to work together on complex projects.

Sprint

In Scrum, a Sprint is like a mini-project that lasts between two to four weeks. It's a set time when the team tries to complete certain tasks.

Think of a Sprint like a cricket match. Just like a team plans how many runs they want to score in an over, a Scrum team plans what tasks they want to complete in a Sprint.

This planning happens in a meeting called a Sprint Planning meeting. The team looks at the list of all tasks (called the Product Backlog), and decides what they can finish in the upcoming Sprint.

During the Sprint, the team meets daily in a quick meeting (called Daily Scrum or Stand-Up) to share updates about what they're working on and if they're facing any problems.

At the end of the Sprint, the team has two meetings. In the Sprint Review, they show what work they've completed in the Sprint to the people interested in the product. In the Sprint Retrospective, the team sits down to discuss what went well and what they can improve for next time.

So, a Sprint is a short, focused burst of activity where the goal is to complete a specific amount of work, and then learn from it to improve for the next time.

Sprint Planning

Sprint Planning is a meeting that happens at the start of each Sprint in Scrum, which is a way of working in software development. This meeting is like the team huddle before a cricket match where the team discusses the game plan.

The team, which includes the Product Owner, Scrum Master, and Development Team, comes together to plan what they are going to do in the upcoming Sprint, which is usually 2-4 weeks long.

They look at the Product Backlog, which is like a long to-do list of all the tasks or 'stories' that need to be done for the product. The Product Owner, who is like the team captain, will have arranged this list in order of priority.

Together, the team selects from this list what they believe they can complete in the upcoming Sprint. They need to be careful not to be too optimistic and overcommit.

Then, they break down the selected tasks into smaller steps or tasks, which makes it easier to get the work done.

By the end of the Sprint Planning, the team should have a clear plan of what they are going to work on for the next few weeks. Everyone in the team should have a good understanding of what they are supposed to do and what the team hopes to achieve by the end of the Sprint.

Daily Standup

A "Daily Standup", also known as a "Daily Scrum Meeting", is a short, time-boxed event (usually about 15 minutes) that occurs each day of the Sprint. It is designed to quickly inform everyone of what's going on across the team. It's not a detailed status meeting, but rather a communication tool.

The structure of the meeting is simple and all team members are expected to attend. They typically stand (hence the name 'standup'), to keep the meeting short and focused.

Each team member answers three questions:

- **What did I contribute yesterday that helped the development team meet the Sprint Goal?** This focuses on individual contributions to the team's progress and encourages accountability.
- **What will I contribute today to help the development team meet the Sprint Goal?** This promotes forward-thinking and helps the team to stay on track.
- **Do I see any blockage that could prevent me or the development team from meeting the Sprint Goal?** This allows the team to foresee potential problems and resolve them before they significantly impact the Sprint's progress.

The purpose is not to provide solutions to issues (that should be done separately), but to surface any issues, blockers, or changes quickly so they can be addressed by the relevant person or people outside of the meeting.

The daily standup is an integral part of Scrum, as it promotes transparency, inspection, and adaptation, the three pillars of Scrum.

Sprint Review

A Sprint Review is a crucial part of the Scrum framework, occurring at the end of each Sprint. This event is a meeting where the Scrum Team and stakeholders inspect the work completed during the Sprint, review the current state of the product, and adjust plans as needed.

The key steps in the Sprint Review include:

- **Work Overview:** The Product Owner goes through the task list, showing what's been completed and what hasn't.
- **Product Demo:** The development team shows what they've done, explains the details of the completed work, and answers any questions. This demo gives a clear picture of the product's current state.
- **Feedback and Discussion:** Stakeholders, who have a vested interest in the product, give their thoughts and suggestions. This feedback shapes future improvements to the product.
- **Forward Planning:** After considering the feedback and the task list, the Scrum Team and stakeholders discuss the next steps. They may add, modify, or prioritize tasks based on their discussions

The purpose of a Sprint Review is to ensure the work being done aligns with the stakeholders' expectations and the organization's goals. Unlike a traditional status meeting, it encourages dialogue and collaboration, with the aim of delivering the most valuable product.

Sprint Retrospective

A Retrospective meeting, or "Sprint Retrospective", is an important aspect of the Scrum framework. It's held at the end of each Sprint, after the Sprint Review, but before the next Sprint Planning.

In a Sprint Retrospective, the Scrum team comes together to reflect on the past Sprint. The team reviews what worked well and what didn't, and decides how to improve moving forward. The goal is continuous improvement.

Typically, the team discusses three key questions:

- **What went well during the Sprint?** This could be anything from great teamwork to successful implementation of a feature.
- **What didn't go so well?** Identifying problems or areas that need improvement can help the team avoid making the same mistakes.
- **What can we improve in the next Sprint?** This includes identifying actionable steps that the team can take to perform better in the next Sprint.

The Retrospective meeting encourages a transparent, open environment for the team to learn from their experiences and continuously improve their processes. This meeting is facilitated by the Scrum Master but all team members are encouraged to participate fully. The team collectively owns their processes and their improvement.

Process of Requirement Analysis in Agile

In the Requirement Analysis phase of Agile methodology, the concept of Epic, User Story, Task, and SubTask play vital roles in defining, understanding, and structuring the work. Let's break it down:

Epic

An Epic is a high-level requirement or a significant feature that needs a substantial amount of work. It is often too large to be accomplished in a single Sprint, so it is broken down into smaller, more manageable parts called User Stories. The Epic helps to provide a broad view of what is to be achieved.

An Epic in the context of an e-commerce application could be "Developing a secure and user-friendly checkout process." This is a high-level work requirement that encapsulates a major area of the application. It is too large to be addressed in a single sprint because it includes various aspects like user interface design, payment gateway integration, handling shipping details, and ensuring the security of transactions.

User Story

During the Requirement Analysis phase, the detailed requirements are broken down into User Stories, which describe a specific functionality or feature from the perspective of a user. A User Story is a simple sentence that captures what a user wants to do with the product, why they want to do it, and what benefit they expect. User Stories are often written using the template: "As a [user role], I want [a feature] so that [a benefit]".

Let's break this Epic down into User Stories:

- **User Story 1:** "As a buyer, I want to review my cart before checkout so I can make sure I've added the right items."
- **User Story 2:** "As a buyer, I want to be able to enter my shipping information easily so that my items are delivered to the correct address."
- **User Story 3:** "As a buyer, I want to select my preferred payment method so I can pay in the way that is most convenient for me."
- **User Story 4:** "As a buyer, I want the checkout process to be secure so that my financial information is protected."

Task

Each User Story can be further broken down into Tasks. These are specific work items or activities that need to be carried out to implement the User Story. Tasks allow the team to divide the work, allocate it among team members, and track progress. For example, for a User Story about adding a new feature to an application, tasks might include designing the feature, coding it, testing it, etc.

Let's continue with the example given in the User Story, tasks should be divided as below.

- User Story 1: "As a buyer, I want to review my cart before checkout so I can make sure I've added the right items."

    - Design the cart review page layout.
    - Develop functionality to display all items in the user's cart, including quantity, price, and subtotal.
    - Implement an option for users to edit the quantity or remove items from the cart.
- User Story 2: "As a buyer, I want to be able to enter my shipping information easily so that my items are delivered to the correct address."
    - Design a user-friendly form for entering shipping information.
    - Develop functionality to save and recall user's shipping information for future purchases.
    - Validate the shipping information entered by the user to ensure accuracy.
- User Story 3: "As a buyer, I want to select my preferred payment method so I can pay in the way that is most convenient for me."
    - Design a payment method selection page with options for various payment types (e.g., credit card, debit card, net banking, wallet payments).
    - Integrate with payment gateways to handle transactions.
    - Develop functionality to save and recall user's preferred payment method for future purchases.
- User Story 4: "As a user, I want the checkout process to be secure so that my financial information is protected."
    - Implement secure protocols for transmission of sensitive data.
    - Integrate with trusted and secure payment gateways.
    - Implement server-side validation of payment details to protect against fraudulent activities.

Sub-Task

If a Task is too complex or requires a lot of effort, it can be broken down into smaller pieces called Sub-Tasks. Sub-Tasks help in making a large task more manageable, and allow multiple team members to work on different aspects of the Task simultaneously.

Let's continue with the task list mentioned in User Story - 1, so how can we divide it into sub-task.

- Task 1: Design the cart review page layout.
    - Identify the key elements to display (product image, name, quantity, price, total price, remove option).
    - Sketch out preliminary designs for the layout of these elements.
    - Create a detailed wireframe for the review page.

    - Obtain feedback on the wireframe from stakeholders and make adjustments as necessary.
- Task 2: Develop functionality to display all items in the user's cart, including quantity, price, and subtotal.
    - Create a database query to fetch all items from a user's cart.
    - Develop a function to calculate the subtotal for each item (quantity x price).
    - Implement functionality to display the fetched data on the review page in the designed format.
- Task 3: Implement an option for users to edit the quantity or remove items from the cart.
    - Design user interface elements for editing quantity and removing items.
    - Develop functionality to update the item quantity in the cart database when a user edits it.
    - Create a function to remove an item from the cart database when a user chooses to remove it.
    - Implement a function to update the display (subtotal and total price) when changes are made to the cart.

So, in the Requirement Analysis phase, you start from the broad view (Epic), narrow down to the user's perspective (User Story), and then get down to the specifics of the work to be done (Task and Sub-Task).

Bugs

Bugs: Bugs refer to coding errors or other problems that cause the software to not work as expected. They are like hitches that need to be fixed for the software to run smoothly. For example, if the website crashes when the user clicks on the navigation bar, that's a bug.

Let's continue with User Story -1, so list of possible bugs should be,

- **Incorrect Item Display:** The items displayed in the cart do not match the items the user believes they added. This might be due to issues in how items are added to the cart or how they are retrieved and displayed.
- **Incorrect Quantity:** The quantity of an item displayed in the cart doesn't match the quantity the user selected.
- **Incorrect Pricing:** The price displayed for items in the cart, the subtotal, or the total might be calculated incorrectly.
- **Unable to Edit Quantity:** The functionality to edit the quantity of an item in the cart is not working.
- **Unable to Remove Items:** The option to remove items from the cart doesn't function as expected.
- **Persistence Issues:** If a user adds items to their cart, leaves the site, and returns later, the cart might not persist and display the previously added items.

- **Performance Issues:** If many items are added to the cart, it might load slowly or not at all.

Suggestions

Suggestions: Suggestions in Agile can come from the team or stakeholders. These are ideas for improvement, whether for the product or the process. These are considered in the backlog grooming or sprint planning sessions and prioritized accordingly.

Let's continue with User Story -1, so list of possible suggestions should be

- **Show Stock Status:** Informing the user of the stock status (In stock, Only few left, Out of stock) of each item in their cart could prevent disappointments or confusion at the checkout stage.
- **Implement Save for Later:** A feature allowing users to move items they don't want to purchase immediately from the cart to a 'Save for Later' list might be helpful.
- **Offer Related Items:** Upon reviewing their cart, users might appreciate suggestions of related items or accessories that complement the products they've chosen.
- **Promotional Offers and Discount Code:** Provide an easy way for users to apply discount codes or gift cards during the cart review phase, before proceeding to checkout.
- **Estimate Shipping Costs:** Give users an estimated shipping cost based on the items in their cart. This could either be a flat estimate or dynamically calculated based on their location.
- **Cart Update Notification:** If an item in the user's cart changes in price or goes out of stock, a notification feature could alert them of this change when they review their cart.

Let's say a QA gives us a suggestion like "Give Offers and Discounts". If a client thinks this is a good suggestion, then we make it a part of our application and implement it. This suggestion then becomes one big task or “Epic” like “Deals and Discount Codes”. From this epic, it will be split into multiple user stories like.

- User Story - 1: As a system admin,. I want a way to control these discount codes, so I can make new ones, change old ones, or stop them when needed.
- User Story - 2: As a customer. I want a place to put discount codes when I am buying things, so I can save money.

So, the big suggestion becomes multiple stories and then can be divided into multiple tasks and subtasks that we can manage and do one by one.

Categories of Tasks and Sub Tasks

In software development, task/subtask categories are used to classify and organize different types of tasks or activities. Here are some common task categories in software development:

- **Backend Development:** Task involves programming and developing the server-side of the application, handling data processing, and managing the logic behind the scenes.
- **Front-end Development**: Task deals with building and implementing the user interface and user interactions on the client-side, visible to users in their web browsers.
- **UI Design:** Task focuses on creating visually appealing and intuitive interfaces, including the layout, color schemes, and overall look of the website or application.
- **UX Design:** Task aims to enhance user satisfaction by improving the usability and accessibility of the product, focusing on user flow and interaction efficiency.
- **Bug Fixes:** Tasks focused on identifying and resolving software defects or issues.
- **Testing and Quality Assurance**: Tasks involving the creation and execution of test cases to ensure software quality and identify any bugs or issues.
- **Documentation:** Tasks related to creating or updating technical documentation, user manuals, or other project documentation.
- **Research:** Tasks aimed at exploring new technologies, frameworks, or methodologies for potential use in the project.
- **Code Review:** Tasks focused on reviewing and providing feedback on code written by other team members to ensure code quality and adherence to coding standards.
- **Deployment:** Tasks related to preparing the software for deployment, including configuration, setup, and release management.
- **Project Management:** Tasks involving project planning, coordination, and tracking progress, such as creating project schedules, managing resources, and conducting meetings.
- **Support:** Tasks focused on providing user support, addressing user inquiries or issues, and maintaining ongoing software operations.

By categorizing tasks, teams can better organize their work, allocate resources effectively, and track progress throughout the software development lifecycle. Also below more categories can be used for detailed bifurcation of tasks or sub-tasks.

- **Infrastructure:** Tasks related to setting up and managing the underlying infrastructure, including servers, networks, databases, and cloud services.
- **Security:** Tasks focused on identifying and addressing potential security vulnerabilities or implementing security measures within the software.
- **Performance Optimization:** Tasks aimed at improving the speed, efficiency, and overall performance of the software application.
- **Integration:** Tasks involving integrating different systems, modules, or third-party services to ensure seamless communication and interoperability.
- **DevOps:** Tasks related to implementing and maintaining continuous integration, deployment, and delivery processes, as well as managing development environments and tools.

- **User Experience (UX):** Tasks focused on enhancing the overall user experience of the software by improving usability, accessibility, and interaction design.
- **Data Management:** Tasks involving data modeling, database design, data migration, or data manipulation within the software application.
- **Code Refactoring:** Tasks aimed at restructuring or improving the existing codebase to enhance maintainability, readability, or performance.
- **Performance Testing:** Tasks involving performance testing, load testing, or stress testing of the software to evaluate its scalability and stability under varying conditions.
- **Technical Debt:** Tasks focused on addressing technical debt, which refers to suboptimal or inefficient code or design choices that may require future refactoring or improvement.

Categories of Bugs

Categorizing bugs aids in prioritizing and resolving issues more efficiently, allocating resources effectively, enhancing communication within the team, tracking performance metrics, and improving overall product quality. Category of bugs should be.

- **Functional:** Bugs related to the incorrect behavior or functionality of a software feature.
- **Blocker:** Critical bugs that completely block or prevent users from performing essential tasks or functions.
- **Grammatical Mistake:** Bugs related to spelling, grammar, or language errors in user interfaces, messages, or content.
- **Functionality Missing:** Bugs where expected functionality is missing or not implemented as intended.
- **Optimization Needed:** Bugs that highlight areas where performance or efficiency improvements are required.
- **User Interface:** Bugs affecting the visual appearance or usability of the software's graphical user interface (GUI).
- **Data Validation:** Bugs related to improper handling or validation of user input or data.
- **Technical Analysis:** Bugs identified during the technical analysis of the software code or architecture.
- **Suggestions:** Non-essential issues that propose enhancements or improvements to the software.
- **Other:** Bugs that do not fit into any of the predefined categories or require further clarification.

Possible Common tasks in Epic/Story

To ensure effective project management, it is important to create an epic or a story for any significant development or change. The responsibility of creating the epic lies with the project coordinator or project lead who directly communicates with the client. They should provide a detailed description of the epic, including any relevant references, and have the option to attach supporting documents if needed.

During the requirements analysis, the project leader or assigned developer is responsible for creating the necessary tasks within the specific story or epic. This ensures that the development process is well-structured and aligned with the overall project objectives.

By following these practices, the project team can effectively manage and track the progress of each development or change initiative. It allows for better communication and understanding among team members, clients, and stakeholders, ultimately leading to the successful delivery of high-quality outcomes.

Requirement Analysis

When new requirements or changes come from the client or business team, it is important to analyze them in detail and convert them into a technical plan. This involves understanding the desired outcome, breaking it down into tasks, and considering technical feasibility and impact. By doing so, the team can effectively plan and execute the project.

The project lead is mainly responsible for requirement analysis but can assign the task to someone skilled in this area. During the analysis process, the assigned person may have questions and can raise them with the business development team, including the project lead.

The project coordinator is responsible for clarifying any doubts for both the assigned person and the project lead.

Spent time for requirement analysis by project lead or project coordinator can be tracked in this task.

Sprint Planning (Task Management)

During the requirements analysis phase, it is crucial to create clear and specific tasks and subtasks by suitable titles with detailed and functional descriptions.

- These tasks and subtasks should cover all the necessary requirements and details for development.
- Each task or subtask should be associated with an epic and user story, establishing a clear connection to the overarching goal.
- To make a better sprint plan it is important to prioritize it and manage dependencies of tasks effectively.
- When estimating the time required for a task, it is essential to ensure that it does not exceed 1 working day length. If a task exceeds this time limit, the task should be divided into multiple sub-tasks under the main task.

- Furthermore, they emphasize the importance of managing task length and breaking down complex tasks to ensure efficient development and timely delivery.

For management of task project lead have to spent time on that and track then spend time under this task

Meeting and Discussion

After completing the requirement analysis, the project lead should thoroughly understand each task and address any uncertainties on a regular basis. To track the hours spent on meetings and discussions, it is recommended to include a common task in all stories or epics.

This task will not be assigned to any specific developer. Instead, it serves as a means for anyone involved in the project to track meeting or discussion hours. The project lead must ensure that each task is understood and resolve any doubts regularly.

To keep track of the time spent in meetings and discussions, every story should have a general task. Any team member involved in the project can use this common task to track the hours spent on meetings and discussions. When tracking time on this task the developer must have to enter specific comments about

- Who has attended a meeting or discussion?
- What was the agenda?
- What was the conclusion?

This approach allows for better tracking and management of meeting and discussion hours by all team members involved in the project. And management can review it by project and make better decisions regarding the ratio of development time and discussion and meeting.

Code Review

Code review is an important phase in the development process because it helps the code better, people learn from each other, make sure everyone follows the same structure and consistency in code, promote teamwork, and result in better software.

After a developer completes their code, another developer or team leader is responsible for reviewing it. Code review should be complete in maximum 3 working days after the task has been completed.

If any team member does not have skills for code review in specific technology, team lead can involve others who have good skills in code review for specific technology.

To make track in this process, the team leader can create a code review task under the story/epic and assign it to a specific code reviewer.

If there are multiple reviewers, there is no need to assign the task to a specific person. The time spent by the reviewer on the code review can be tracked under this task.

Task Verification

After a task is completed by the team, it is important for the project lead to verify whether the task has been completed as per expectations. This approach reduces the need for excessive collaboration between the QA and development teams, especially for minor, easily detectable bugs.

Ultimately, this helps save time for everyone involved and reduces costs for the company. The project lead can track the time spent on testing tasks that have been completed by the team under this specific task.

Code Deployment

To deploy code for a task or any previously undeployed tasks, the project lead can create a deployment task and assign it to the respective teammate who is responsible for deploying the code on the server. He/she will need to spend time on source code deployment, server configuration, and uploading sample data to the testing or live server. They have to track spend time for deployment under this task.

Their responsibility involves the creation of a detailed deployment task, which is then assigned to a designated team member. This team member is tasked with managing the server-side deployment of the code.

QA Testing

Once a feature has been developed and all the related tasks are finished, it undergoes verification by the project lead before moving to the testing phase.

During sprint planning, specific tasks for QA testing should be created and labeled with a unique tag or ID. These tasks should list the stories and tasks that will be tested by the QA team.

Task title should be “Testing - <UAT release ID>”. Description should be a list of all tasks or stories which need to be tested under this release.

Furthermore, the tasks should include information about when the testing will start and how much time will be spent for testing the developed features or modifications.

Technical Documentation

Technical documentation serves a vital function in the development process, acting as a valuable reference for future scenarios such as onboarding new team members, transferring work to another team member, among others.

Upon the completion of all planned tasks and after conducting a code review, it is imperative for the assigned developer(s) to create or update technical documents, providing a comprehensive overview of the developed features and functionalities.

To streamline this process, a task dedicated to technical documentation should be created under the corresponding story.

The Project Lead is responsible for creating a specific task within the story for the documentation of any developments or modifications made to the application.

Once a specific feature or set of task development is completed with review of Project Lead, developers must have to write or change technical documents.

Task list and Task Transition

Backlogs (Common for all Sprint)

A product backlog is a list of important things to do for a product or project. It is managed by the product owner/leader and includes all the tasks needed to make the product successful. The items in the backlog are usually described as user stories and may change as new requirements come in and priorities change.

Based on the requirements, dependencies, and priorities, specific backlog items are turned into stories or tasks. These are then moved to the to-do list for the current or next sprint, making sure they are addressed promptly and efficiently.

As per our defined process, the task list on Jira should be as mentioned below. Sprint wise task list should be–

1. To do
2. In Progress
3. In Code Review
4. Development Completed
5. QA Review Pending (Ready to UAT Release)
6. Bug Found
7. QA Completed (Ready To Live Release)

Let's understand the work process and task transition for each task list in a better way.

1. To do

In agile software development, a “To do” list is a list of tasks or user stories that a team plans to complete during a sprint. It guides the team's daily work and represents their commitments for the sprint.

During the sprint planning meeting, the team decides which user stories or tasks to include in the to-do list. Each item on the list should be clear and specific, so everyone understands what needs to be done.

After discussing with the client, the team creates a sprint plan based on task priorities and technical dependencies. They move tasks from the product backlog to the sprint backlog or create new tasks in the to-do list. These tasks have titles, detailed descriptions, assigned team members, estimated time, due dates, and appropriate labels.

It's important to make sure the sprint plan is realistic and achievable within the sprint timeframe. The team should consider any risks or obstacles that may affect task completion and address them. The sprint plan is reviewed and adjusted as needed during the sprint to stay on track and achieve the sprint goal.

2. In Progress

During sprint planning, it's important for team members to prioritize tasks and start working on them with full concentration. Before beginning work, the team member should move the task to the "In Progress" list.

Task Transition after Task complete

Once the development of a task is finished, it should be moved to one of the following lists, depending on the task or project nature:

- Code Review – If it needs to be reviewed by other team members or seniors to ensure quality.
- Development Completed – If the task is completed without needing further code reviews.
- In QA Review – If the task requires review by the QA team before it can be marked as completed.

Choosing the right list ensures that the completed task undergoes thorough review and testing before it is considered done. This helps in maintaining the quality of the work delivered.

If task not complete on day end

- Track the time spent on the task on day with a comment for time spent on task.
- Write clear and concise comments about what was completed and what remains to be done.
- Communicate any obstacles or challenges faced during development to ensure successful completion of the task.

After task complete

- Track the time spent on the task on that day with a comment for time spent on task.
- If the task is related to development, write a descriptive commit message “#<Task-ID> - <message>” and push the code to a specific branch on Git.
- Add a comprehensive comment on the task to explain the affected modules and where they can be found in the application. Also include screenshots or recorded videos if necessary.
- Move the task to the respective list with a meaningful task comment.

3. In Code Review

Once the development of a task is finished, it should be moved to the code review stage. Usually, code reviews are done by team members from the same or different teams, but the project leader decides who will review a specific code.

By reviewer, ideally code review should be completed within 2 working days after task completed

If any improvement required,

- Task will be moved back to the "To do" list.
- Please add required comments in the task list.
- By developer, improvement should be completed within 2 working days after the task moves back to the To do list.
- After improvement done –
    - Time spent on improvement should be tracked on the same task only.
    - Move the task into the “In Code Review” list and update about changes to the reviewer.
    - Reviewer will review the changes again.

If no changes are required, it will be moved to the "Development Completed" list.

4. Development Completed

Once tasks are completed and the code review is done, the reviewer will be listed in “Development Completed”.

The project lead is now responsible for verifying that the task has been developed according to the task description and meets all expectations and objectives as defined in the task scope. The project lead plays a crucial role in confirming the quality and completeness of the work done before it is deemed ready for further stages or delivery.

If the task does not work as expected, the project lead will add a comment to the task and move it back to the "To-do" list. The assignee must resolve all the issues in the task according to the timeline suggested by the project lead. If necessary, the project lead may also create a bug report for the same issue. When a bug is created, the QA bug creation process guideline must be followed, and the assignee must resolve the bug in the same release that was planned.

After task testing and verification done by the project lead, they will be responsible for deploying code on UAT and move tasks on “QA Review Pending” list.

6. QA Review Pending (Ready To UAT Release)

The QA team is responsible for testing all the tasks and user stories listed in their testing ticket. They can start testing according to a schedule made into sprint planning or testing of specific RC (Release Candidate) version. If they found any bugs or issues during the testing process,

- QA will move the task or user story to the 'Bug Found' list and appropriately label it.
- A bug ticket will be created in Jira, referencing the original task. If QA cannot find an exact task reference for a specific bug, they can refer to a related user story instead.
- A bug ticket will then be moved to the "Todo" list.

QA has to create a testing task for subsequent versions of the release candidate. And add newly identified bugs in the newly created testing task. Discuss with project lead for schedule testing of newly created testing tasks.

Once QA confirms that a task or user story meets the expected requirements, it will be marked as “QA Completed”.

During the testing of a particular task, if QA notices any improvements, additions, or changes to a feature related to that task, they can discuss it with the project lead and create an issue ticket as a "Suggestion". After discussing and reaching a conclusion–

- If the suggestion aligns closely with the scope of the task or story, it will be moved to the Todo list for the current sprint.
- However, if the suggestion is beyond the scope or particularly noteworthy, it will be moved to the backlog.

For suggestions in the backlog, the business team or project coordinator will engage in discussions with the client. Once the client approves a suggestion, it will be converted into a task or user stories or epic as discussed earlier. Otherwise suggestions will be ignored and make it close.

6. Bug Found

All tasks or stories that have bugs or issues will be listed on this task list. The assigned developer is responsible for fixing these bugs or issues listed in the “To-do” list within 2 working days.

If resolving a bug will take a significant amount of time, it is necessary to discuss this with the team leader. The team leader will make a decision timeline of bug resolution, and the assigned person will then work on fixing the bug accordingly. After all bugs resolved of specific release candidates.

7. QA Completed (Ready To Live Release)

All tasks which are completed and tested by QA will be listed in this task list. After making the live release of specific tasks, those tasks will be marked as completed by the Team leader.

Task Transition for User Story

When development begins on a user story, that story should be shifted to the "Todo" list for the specific sprint, and the sprint number should be labeled. If the story extends over more than one sprint, multiple sprint number labels should be added. The task should then be moved to the "Todo" list for the next sprint. Once all development tasks or user stories are launched in production, they should all be marked as completed.

Remember that, In certain exceptional cases, for client-specific projects where QA is not necessary from our end, clients may prefer to assign tasks on a daily basis without requiring documentation or code review. In such situations, after discussing with the project manager or coordinator, the project or team leader must adapt and manage a task list and work process that best suits the nature of the project, rather than strictly adhering to our predefined task list and work process. This flexibility allows us to cater to the unique requirements and preferences of each client and project.

Sprint and Task Planning Process

Manage Backlog

Managing backlogs in Agile methodology involves

- Creating a prioritized product backlog: Develop a list of project items in order of priority.
- Refining user stories: Fine-tune and break down user stories into actionable tasks.
- Prioritizing items based on value and urgency: Determine the importance and time-sensitivity of backlog items.
- Planning sprints with selected backlog items: Choose specific backlog items for each sprint cycle.
- Conducting daily standups to track progress: Hold brief meetings to update the team on daily progress and challenges.
- Regularly reviewing and updating the backlog: Review and adjust the backlog regularly to keep it relevant and up to date.
- Prioritizing based on changing needs: Adjust the priority of backlog items as project requirements evolve.
- Holding sprint reviews: Conduct meetings to showcase completed work and gather feedback from stakeholders.
- Retrospectives for feedback and improvement: Reflect on the sprint's outcomes, gather feedback, and identify areas for improvement.

This ensures that the team focuses on valuable work, adapts to changes, and continuously improves backlog management.

Responsibility for adding more work to the backlogs lies primarily with the product owner in collaboration with the project team and stakeholders. The product owner is responsible for managing the backlog and making decisions regarding the inclusion and prioritization of work items. However, effective collaboration and communication among all project members are essential for successfully incorporating additional work into the backlog.

Sprint Planning

Once the backlog is created, the tasks are analyzed by a project leader or a specific developer based on their skill set and efficiency. Project lead have to assess below things for make better sprint planning

- **Clear Sprint Goals:** Ensure that the sprint goals are well-defined and understood by the entire team. This clarity helps guide the planning process and keeps everyone aligned on what needs to be accomplished.
- **Prioritize the Backlog:** Collaborate with the product owner to prioritize the backlog items based on value, urgency, and dependencies. This helps the team focus on the most important and feasible items during sprint planning.

- **Refine User Stories:** Prior to sprint planning, invest time in refining user stories. Break them down into smaller, actionable tasks, and ensure that they have clear acceptance criteria. This preparation allows for more accurate estimation and planning during the session.
- **Capacity Assessment:** Evaluate the team's capacity and availability for the upcoming sprint. Consider factors such as team size, individual availability, and any planned leave or commitments. This assessment ensures that the planned workload aligns with the team's capacity.
- **Time Boxing:** Set a time limit for the sprint planning session to keep it focused and avoid excessive discussions. This encourages the team to prioritize and make decisions efficiently. Ideal timebox for sprint should be 2 to 3 weeks.
- **Involve the Entire Team:** Ensure that all team members actively participate in sprint planning. This includes developers, testers, designers, and other relevant stakeholders. Their input and perspectives contribute to better decision-making and more accurate planning.
- **Break Down and Estimate Tasks:** Collaboratively break down user stories into smaller tasks during the planning session. Estimate the effort required for each task using techniques like planning poker or relative sizing or skills. This helps in better capacity allocation and workload distribution. Project lead can involve the team to estimate assigned tasks and verify estimation.
- **Define Acceptance Criteria:** Clearly define acceptance criteria for each user story and task. This ensures a shared understanding of what constitutes a completed item and helps prevent misunderstandings during development.
- **Resolve Ambiguities:** Address any questions or uncertainties during the planning session. Encourage open discussions and involve the product owner to provide clarifications and resolve any ambiguities that may impact the planning process.
- **Review and Adapt:** After completing sprint planning, take a moment to review the planning session itself. Identify any areas for improvement and make necessary adjustments to optimize future planning sessions.

Sprint planning should be completed before 2 working days of the sprint start date by project lead and then provide a plan to internal stakeholders.

Task Assignment and Estimation

During the sprint planning phase, the assignee of each task takes the responsibility of estimating the hours required to complete it. This assignee is typically a developer or a team member who will be actively working on the task. They utilize their expertise and experience to estimate the effort involved accurately.

After the assignee provides the initial estimation, the team leader verifies and reviews the estimation. The team leader plays a crucial role in ensuring that the estimation aligns with the task requirements and overall project goals. If the team leader feels that the estimation is not appropriate or if there are any discrepancies, they engage in discussions with the assignee to reassess the estimation hours.

It is important for the team leader to consider various aspects when evaluating the estimation. They need to ensure that the estimated hours cover not only the development phase but also additional tasks such as code optimization, testing, and quality assurance.

This includes testing the task against all possible test cases and examining its impact on other areas of the system. The team leader's expertise helps in striking a balance between delivering quality work and meeting the project's timeline.

When Subtask Required?

When a task requires multiple skill sets, technologies, or human resources, it is often necessary to break down the task into subtasks. This can help ensure that each component of the task is completed in the correct sequence, and that the overall task is completed efficiently and effectively.

Creating subtasks can also help with proper dependency management, which is crucial in ensuring that the task is completed without any delays. By properly managing dependencies between subtasks, team members can be made aware of which tasks need to be completed before they can begin work on their own tasks.

Breaking down a task into smaller, more manageable subtasks also makes it easier to assign responsibilities and track progress. Each subtask can be assigned to a specific team member, based on their skillset and availability. This can help ensure that team members are working on tasks that are appropriate for their expertise and that the overall task is completed as quickly as possible.

Insufficient Backlog? What to do?

The project lead plays a crucial role in ensuring that the entire project team remains engaged and productive. If there is an insufficient backlog of tasks to plan for a sprint, To address the situation of insufficient backlogs in a sprint, consider the following actions:

- **Shorten Sprint Duration:** Consider reducing the time box of the sprint to one week and communicate this update to stakeholders and resource managers. This allows for more frequent iterations and adjustments.
- **Conduct Research and Development:** Utilize the available time to conduct research and development activities related to future work. This can involve exploring new technologies, frameworks, or approaches that can enhance the project's progress.
- **Code Improvements:** Assess the current codebase and identify areas of improvement. Address any code inefficiencies, bugs, or performance issues to enhance the overall quality and maintainability of the software.
- **Security Enhancements:** Pay attention to the security aspects of the project. Perform security assessments, address vulnerabilities, and implement necessary measures to strengthen the system's security posture.
- **Framework or Technology Upgrades:** Evaluate if any framework or technology used in the project requires updates to newer versions. This ensures compatibility, performance enhancements, and access to the latest features.
- **Documentation Improvements:** Review and enhance project documentation, including technical documentation, user guides, and API references. Improve clarity, completeness, and accessibility to facilitate understanding and future development.
- **Resource Reallocation:** If the team size is larger than necessary for the current workload, consider reallocating resources to other projects where they can be utilized

more effectively. This optimizes resource utilization and balances workload across teams.

By implementing these strategies, project leads can make productive use of the available time, address areas for improvement, and continue progressing the project even with limited backlogs.

When Estimation and Due Date change?

Estimations and due dates can be modified under the following circumstances:

- **Change in task requirements:** If the client or project co-ordinator alters the task requirements after it has been estimated during sprint planning, the estimation and due date can be adjusted accordingly.
- **Urgent tasks or change requests:** If an urgent task or change request is received that affects a planned task, the estimation can be modified to accommodate the new priority.
- **Delays in dependent tasks:** If a dependent task is delayed by another developer, the due date may need to be changed to account for the delay and maintain the project timeline.
- **Research tasks with valid reasons:** Research and development tasks often involve exploring new, learning, experimenting, or conducting in-depth analysis based on scope of research. If there is a solid reason, such as unexpected complexities or the need for additional research, it is acceptable to modify the estimation or due date for R&D tasks. This ensures that the necessary time and effort are allocated for thorough exploration and development.
- **Extended task duration:** When a particular task takes longer than expected, the due date of all dependent or subsequent tasks may be modified to reflect the revised timeline.

By allowing flexibility in these situations, project teams can adapt to changes, ensure efficient planning, and meet project objectives. Effective communication with stakeholders is essential to keep everyone informed about any adjustments made to estimations and due dates.

When Assignee Will be Changed in Task?

Once work begins on a task and it is assigned to a developer or designer, we generally aim to keep the same person working on it throughout. However, there are some cases where the project lead may need to change the assignee:

- **High priority and unavailability:** If a task is extremely important and the originally assigned person is unavailable, the project lead may change the assignee. This ensures that the task gets immediate attention and progress is not delayed.
- **Emergency or extended leave:** If the assignee is on emergency leave or has an extended absence, the project lead may need to change the assignee to ensure the task doesn't get delayed. This helps in effectively managing resources and maintaining project continuity.
- **Skill or expertise requirements:** If a task requires specific skills or expertise that the original assignee does not possess, the project lead may change the assignee to

someone who has the necessary abilities. This ensures that the task is handled effectively and meets the required quality standards.

- **Balancing workload:** If a team member is burdened with too many tasks or has reached their maximum capacity, the project lead may redistribute the workload by assigning some tasks to other team members. This helps prevent exhaustion, maintains productivity, and ensures timely completion of all tasks.

It is important for the Team Leader to assess each situation and make informed decisions when changing the task assignee. Effective communication and coordination with the team members involved help ensure a smooth transition and continued progress on the project.

By making these necessary changes, the project lead ensures that tasks are managed well, resources are used efficiently, and the project stays on track. It is important for the project lead to communicate these changes clearly to all relevant team members, so everyone is aware of the new assignee and any impact on timelines or deliverables.

Managing Release Version

The Release Version Format

The practice of separating release versions by dots (e.g., v1.0.0) is a common convention used in version numbering schemes. The dot notation allows for a structured representation of the version number and helps convey specific information about the release.

Here's a breakdown of the typical meaning behind each component separated by dots:

- **Major Version (1):** The major version indicates significant changes to the software, often introducing new features or architectural modifications. Incrementing the major version number suggests that the update may not be fully backward compatible with previous versions.

- **Minor Version (0):** The minor version represents smaller enhancements, improvements, or additions to the software. These updates typically maintain backward compatibility with the previous major version.

- **Patch Version (0):** The patch version is used for bug fixes, security patches, or other minor updates that don't introduce new features or alter the existing functionality. Patch versions generally maintain both backward and forward compatibility.

By separating the version number into these distinct components, it becomes easier to understand the scope and impact of a particular release. Users can quickly identify whether a new version is a major update, a minor enhancement, or a simple bug fix.

While the specific format of version numbers can vary across organizations and projects, the dot-separated numbering scheme is widely adopted and helps provide clarity and consistency when managing release versions.

Effective Git Management in Project

To effectively manage release versions for UAT and production, you should follow these steps:

Branch Setup in Git and Development Workflow

- Maintain three branches in your Git repository: "development," "UAT," and "master/main."
- Developers actively work on the development branch, creating separate branches for their specific features or modules.
- Developers complete their feature/module development on their respective branches and merge their changes with the development branch.
- Regular code reviews and quality checks should be conducted to ensure the code meets the required standards.
- The project lead or designated team verifies the code, features, and functionality developed by the team on the development branch.
- Once validated and verified, the development branch is merged into the UAT branch in preparation for UAT testing.

UAT Testing and Validation

- The UAT branch is active in the UAT environment where QA testers thoroughly test all the features.
- QA testers validate the functionality, perform regression testing, and ensure that all requirements are met.
- Any identified issues or bugs are reported and addressed by the development team on the development branch.

Production Deployment

- After successful UAT testing and resolution of any identified issues, the UAT branch is merged into the master/main branch.
- This merge signifies that the code has passed UAT and is ready for deployment to the production environment.
- The master/main branch contains the stable and approved code for production.
- Deployment to the production environment is performed using the code from the master/main branch.

Tag Version management in Git

In each Release either Production or UAT deployment must have to create a tag in Git with release ID. Also must have to mention below detail in description

- **List of new or affected Features:** Provide a comprehensive list of the features included in the UAT release. This ensures that all stakeholders are aware of the scope and functionalities being tested.
- **List of PMS Tasks and Change logs:** Include a list of associated PMS (Project Management System) tasks with task id.

- **Known Issues or QA Notes:** Communicate any known issues that should be ignored during the QA process. Additionally, include specific notes or instructions for the QA team to follow during testing.

Follow the organization's established deployment process, including proper backups, monitoring, and rollback plans.

By following these steps, you can ensure a structured and controlled release management process, maintaining separate branches for development, UAT testing, and production. This approach allows for thorough testing, issue resolution, and controlled deployment to the production environment, minimizing risks and ensuring the stability of your software releases.

UAT & Production Release Version Management

In the production server the first version should be v1.0.0. So In UAT it should be known as a release candidate (RC1) for testing phase of application it will be known as v1.0.0-rc1.0. So the steps should be

UAT Version Tag in Git and Testing

- Establish the initial production version as v1.0.0.
- Create a release candidate (RC1) from the development branch, referred to as v1.0.0-rc1.0, and merge it into the UAT branch.
- Deploy the UAT branch's code into the UAT environment for thorough testing and validation.
- Conduct comprehensive testing to identify any bugs or issues.

Bug Fixes and Subsequent Release Candidates

If bugs are discovered during testing:

- Resolve the identified bugs with the assistance of the development team.
- Create a new sub-release candidate, v1.0.0-rc1.1, incorporating the bug fixes.
- Conduct regression testing on the updated release candidate in the UAT environment.
- If the release candidate successfully passes validation, merge v1.0.0-rc1.1 into the master/main branch.
- Create a version tag, v1.0.0, to represent the final approved version.
- Deploy the code from the master/main branch to the live environment.

Live Deployment

- If no bugs are found during the testing phase, proceed as follows:
- Create a version tag, v1.0.0, from the UAT branch and merge it into the master/main branch.
- Deploy the code from the master/main branch to the live environment.

In this example, the release candidate versions follow the format "x.y.z-rcn," where "x.y.z" represents the final release version, and "n" indicates the release candidate number. Once deploy on production , the release candidate becomes the final live release version by dropping the "-rc{n}" suffix. This versioning scheme provides clarity on the progression from release candidates to the final approved version for deployment.

Using these steps, you can maintain a structured release management process, starting with the initial version, creating release candidates for testing and bug resolution, and ultimately deploying the approved code to the live environment. This approach helps ensure a controlled and systematic progression from UAT to production, with proper versioning and tagging at each stage.

Release/Deployment Process

Maintain Single or Multiple Release in UAT Server

In the software development process, it is common to have multiple rounds of Quality Assurance (QA) testing between releases on a live server. To manage these testing phases effectively, you can create multiple tasks dedicated to QA testing in your project management tool, such as Jira.

Each QA testing task should specify a subset of the upcoming live release version. For instance, let's say the next live release version is tagged as v1.2.0 In this case, based on task dependency, nature and complexity you have divided testing phase in 3 parts then created three QA testing tasks with the IDs

- First release should be v1.2.0-rc1.0,
- Second release should be 1.2.0-rc2.0,
- And the third release should be 1.2.0-rc3.0.

This ID should be used in the task title also. UAT deployment tasks should use the same ID for each release in UAT.

Now, let's assume that during the testing of version 1.2.0-rc2.0, some bugs are identified. If you want to release the application and retest it after resolving the bugs, you can create a new tag ID, such as 1.2.0-rc2.1. This indicates that it is a subsequent testing phase for the same version. If further bugs are found in the same version and resolved, you can release it with an incremented tag ID, such as 1.2.0-rc2.2.

The advantage of organizing QA testing tasks in this manner is that it allows for clear identification and tracking of different testing phases within a specific version or subversion. Each task represents a specific testing effort, and any bugs or issues discovered during testing can be associated with the respective task.

This approach provides a structured framework for QA teams to conduct testing, report issues, and retest after bug fixes. It helps in maintaining version control and ensures that all identified issues are appropriately addressed before a release.

By utilizing this tagging system for QA testing tasks, you can effectively manage multiple testing phases within a live release version, facilitating a systematic and controlled approach to software testing and ensuring the delivery of a high-quality application.

Release Process in UAT

As per the sprint plan, the project lead and project coordinator have decided to deploy or release features on the User Acceptance Testing (UAT) environment with a specific version number or ID. To meet this timeline, the project lead must have to ensure that all tasks are completed by the development team. This involves code commenting, code review, and task testing by both the developer and the project lead.

In cases where a person is not involved in testing, the project lead has the flexibility to move the task after the completion of code review. If a task involves minimal changes that do not affect the application flow, such as text or color modifications, the project lead can directly move it to "QA Review Completed."

Before the QA testing process begins, it is important to verify that all tasks have been completed according to the plan. If there are any minor issues, it should be discussed and resolved with the QA team and project coordinator. The project lead can proceed with further processes only after obtaining their approval. Otherwise, the issues must be fixed before proceeding.

During the UAT release, it is essential to follow a well-defined process to ensure efficient testing and bug resolution. Here are the steps that should be taken during the UAT release:

Perform Deployment and Data Setup: Start by deploying the UAT release, ensuring that all required testing data is in place. This step sets up the necessary environment for conducting thorough testing.

Task Transition for UAT Testing: Move all tasks related to the UAT release from the "Development Completed" list to the "QA Review Pending" list. This transition clearly indicates that the tasks are ready for QA review and testing.

Notify UAT Release: Notify the relevant stakeholders about the UAT release by providing the following information:

- **UAT Release Version Number:** Clearly state the version number assigned to the UAT release (e.g., v1.2.0-rc1.0). This helps in tracking and identifying the specific release being tested.
- **Uploader's Name:** Mention the names of the individuals responsible for uploading the release. This ensures accountability and provides contact points for further communication.
- **Verifier's Name:** Mention the names of the individuals responsible for verifying the release.
- **Last GIT Commit ID:** Include the unique identification number of the last GIT commit associated with the UAT release. This promotes traceability and version control.
- **List of Included Features:** Provide a comprehensive list of the features included in the UAT release. This ensures that all stakeholders are aware of the scope and functionalities being tested.
- **List of PMS Tasks:** Include a detailed list of associated PMS (Project Management System) tasks, including task numbers, titles, assignee names, and links to the tasks. This facilitates easy access and reference during testing.
- **Known Issues or QA Notes:** Communicate any known issues that should be ignored during the QA process. Additionally, include specific notes or instructions for the QA team to follow during testing.

Identified Bug Resolution: If the QA team identifies any bugs during testing, they should be resolved in allotted reserved time for bug fixes. It is important to track the time spent on resolving these bugs by logging it on the respective bug issues.

Bug Resolving Process

After the UAT release, the QA team will thoroughly test all the tasks and features mentioned in the UAT release mail, following the sprint plan. Project lead can plan multiple releases for one or more stories, or they can be combined into a single UAT release. However, it is important to have a well-defined and fixed plan for these releases.

Time reserved for bug Fix after releasing it in UAT

As per the QA plan, the QA team will test all the released features and functionalities. If any bugs are discovered during testing,

- The QA team will create bug issues and mark the corresponding tasks as bugs.
- During sprint planning, the project lead must allocate time for bug fixing buffer time based on the QA plan either in the same sprint or next sprint.
- Project lead can reserve up to a maximum of 15% of the total hours for bug fixing, considering the complexity of the project modules or tasks.
    - Let's assume, based on UAT version task development complexity, suppose project lead has set 10% buffer time.
    - For example, if there are 15 tasks released in the UAT version and 120 hours were spent on those tasks,
    - Then 12 hours (10% of 120 hours) should be reserved for bug fixing in the same sprint or the next sprint.
    - Once a bug is opened or created by QA, it should be fixed on that planned duration.
    - If any fixed bugs are reopened, the same rule applies, and hours should be reserved for bug fixing. For example, if 10 hours were spent fixing some bugs, then 1 hour should be reserved for fixing reopened bugs.
- In case of extra effort required to fix bugs, project lead and developers may need to put in additional hours by giving extra time.
- It is important for developers to discuss such situations with the project lead and project coordinator, as unexpected and abnormal circumstances may arise when dealing with bug issues.

When planning a story in a particular sprint, the project lead, developer, and QA team must discuss and set a due date for the story to go live. The team leader has flexibility in deciding the flow of development and the UAT release process.

Post-Bug Resolution Steps

Move Tasks and Bugs to "QA Review Pending": After resolving the bugs, move all associated tasks and the bugs themselves to the "QA Review Pending" list. This ensures that the updated code and fixes are reviewed before further testing.

Notify UAT Sub-Release: Notify the UAT sub-release with the UAT release sub-version number (e.g., v1.2.8.1, v1.2.8.2.. so on). This communicates that a new sub-version has been created after bug resolution.

Reopened Issues: If any issues are reopened within the same release, they should be fixed on the same day. After fixing them, notify the UAT sub-release with the next sub-version number (e.g., v1.2.8.2.1). This ensures prompt resolution and effective communication within the release cycle.

By following these steps, you establish a structured and transparent process for UAT release, testing, bug resolution, and version management. This promotes efficiency, accountability, and effective collaboration among team members involved in the UAT phase.

When released in Live without QA Testing?

There are situations where we may need to release the software live without QA testing for demo purposes or other specific circumstances. Some of these cases include:

- **Client's urgent demo request:** If the client needs a quick demonstration of the software, there might not be enough time for full QA testing. In such cases, we can mention in the release email that the live release is being done solely for demo purposes, with the understanding that thorough testing will follow.
- **Unavailability of QA team:** If the QA team is unavailable on the scheduled release date due to unforeseen circumstances or limited resources, conducting comprehensive testing may not be possible. In the release email, we can explain that the live release is happening without QA testing due to the team's unavailability and ensure that testing will be performed later.
- **Hotfix or critical issue:** In urgent situations where a critical bug or issue is affecting the software's functionality or security, it may be necessary to release a hotfix or patch immediately without going through the standard QA process. The focus is on resolving the urgent issue to restore normal operations.
- **Rollback and re-release:** If a previous version of the software had already undergone thorough QA testing and was released successfully, but a subsequent release introduced a significant issue, it may be decided to rollback to the previously tested version and re-release it without conducting QA testing again.
- **Limited functionality or feature release:** In certain cases, a release may involve only a limited set of functionalities or features that have been thoroughly tested and are not expected to impact the overall stability of the system. This approach allows for the gradual release of specific components or modules while ensuring that critical areas have been tested adequately.
- **Client's desire to go live:** Sometimes, clients may have an urgent need or preference to go live with the software, even without completing full QA testing. This could be due to business requirements or time constraints. In such cases, we can mention in the release email that the live release is happening based on the client's

request, with a commitment to conduct further testing and address any issues promptly.

It is important to communicate these reasons clearly in the release email, ensuring that everyone is aware of the decision to release without QA testing. This sets the expectation that subsequent testing and bug fixing will be prioritized to ensure the software's quality and stability.

Proper documentation and necessary approvals from relevant stakeholders should be obtained before proceeding with the live release.

Release Process in Production

If the QA team is involved in the project, their approval is required before proceeding further to deploy the version on production. Once approval is obtained from the stakeholders including client, and the release is uploaded on the production environment, the project lead have to take the following steps:

- **Live release mail:** The project lead sends a live release mail to all project members, sharing the live release version number. This ensures that everyone is informed about the new release.
- **Completed tasks:** The project lead marks all the released tasks as completed in the Jira. This helps track the progress and completion status of the tasks.
- **Git Tag creation:** A tag is created on Git to represent the released version. This allows for easy reference and identification of the specific version of the code.
- **Branch merging:** The code branch that contains the released version is merged into the Master/Main branch. This ensures that the latest code is integrated into the main branch for future development and releases.

After the code is released in the production environment, the project lead sends a live release mail with the following details:

- **Live release version number:** Specifies the version number of the live release (e.g., v1.2.0).
- **Deployed by:** Mentions who uploaded the code to the live environment.
- **Verified by:** Indicates the team member responsible for verifying the release.
- **Tested by:** Refers to the person who conducted testing on the release.
- **Approved by:** Specifies the person who provided approval for the release.
- **Last GIT commit ID:** Mentions the unique identifier of the latest Git commit associated with the release.
- **List of Features:** Provides a comprehensive list of the new features or enhancements included in the release.
- **List of PMS tasks:** Specifies the jira tasks associated with the release, including their task number, title, assignee, and link.
- **Known issues:** Highlights any known issues or bugs that are present in the released version.

By sharing this information in the live release mail, the team ensures that all project members and stakeholders are informed about the details of the release, allowing for transparency and effective communication.

Why Technical Documentation is Required?

In software development, it's crucial to keep clear and complete records of our work. Let me explain why technical documentation is so important and how we handle situations when we can't update it right away:

The Importance of Technical Documentation

Technical documentation is like a guidebook for our projects. It helps developers, project stakeholders, and future team members understand the code and system. After finishing each task, developers should update the documentation right away. This keeps it accurate and up-to-date, reflecting what's actually in the code. By doing this, we can track our progress and keep everything consistent throughout the project.

Challenges and Solutions

Though we prefer updating documentation immediately, sometimes we face challenges. Urgent client requests or high-priority work might come up, leaving no time for immediate updates. In such cases, updating the documentation right away may not be possible.

To handle this, we've set a reasonable time limit. Developers must complete the documentation within 15 days after finishing a task. This gives them time to focus on urgent work while still making sure the documentation doesn't get ignored.

Project Lead's Role

Our project lead takes responsibility for overseeing the completion of technical documentation within this timeframe. They'll check on the status of the documentation and follow up with developers to make sure it's done promptly. This way, we can keep the documentation accurate and consistent for future reference and project maintenance.

Benefits of Timely Technical Documentation

Updating technical documentation on time brings several benefits:

- **Easy to write:** If a developer updates the document right after completing each functionality or task, it becomes much easier to remember what needs to be written in the document.
- **Knowledge Transfer:** Good documentation helps team members share knowledge, making it easier for everyone to understand the project.
- **Easy Onboarding:** New team members can quickly get familiar with the code and project structure through clear documentation.
- **Less Technical Debt:** Well-documented code and system architecture reduce technical debt and improve code quality, leading to fewer issues in the long run. Clear documentation helps developers understand and maintain the code easily, ensuring a smoother and more efficient development process.

- **Efficient Updates:** When the documentation matches the code, developers can easily update and improve the project.

By stressing the importance of technical documentation and giving a reasonable timeframe for updates, we ensure high-quality work, good teamwork, and efficiency throughout the development process.

Checklist for Developer and Designer

All developers and designers have to follow the below checklist for each and every task development.

While Creating Task

Before starting a sprint, it is important to plan it in a well-organized manner. This involves creating tasks that are properly planned and organized for the team. Creating tasks is a crucial part of sprint planning, and project leads must take certain points into consideration when creating new tasks.. Here are some important considerations to keep in mind when creating a task:

- **Create a meaningful and appropriate task title:** The title of the task should be concise and accurately reflect the task's objective and scope.
- **Write a clear and concise task description:** The task description should cover all necessary details and requirements and be easy to understand and follow. Make sure to analyze the client's requirements thoroughly to ensure that all aspects of the task are accounted for.
- **Apply task tags properly:** Use appropriate tags to ensure easy categorization and retrieval of the task.
- **Mention task category:** Use appropriate task category from pre defined task category list.
- **Attach all required files and reference links:** If any required files or reference links are needed to complete the task, they should be properly attached to the task.
- **Assign the task based on the skillset:** Assign the task to a team member who has the necessary skills to complete it efficiently and effectively.
- **Confirm task dependencies:** If the task is dependent on other tasks, it is important to confirm the due date of those tasks with the assignee to ensure that there are no delays in completing the task. Use the dependency feature in the project management system to ensure proper tracking and management of tasks.
- **UAT Release Version:** In order to streamline the deployment process in the UAT environment, it is essential to set the UATrelease version for each specific task. This enables smoother coordination for subsequent stages such as UAT release, QA process more efficiently with effective work flow..
- **Live Release Version:** To optimize the deployment process in a live or production environment, it is crucial to assign a Live release version to each specific task. This practice ensures a more seamless plan for live release, enhancing the overall workflow and efficiency of the deployment process.

By following these steps, you can ensure that tasks are created effectively and efficiently, helping to ensure the success of the project.

Before Start Work on Task

To ensure the success of a project and effective task completion, it's essential to follow a set of guidelines. Here are the steps that should be taken prior to start work on any task:

- Have you ensured all the details, fields, category and tags, release versions in the task mentioned properly and not missed anything?
- Assigned task is associated with the current sprint?
- Have you ensured that task descriptions are detailed and descriptive?
- Have you verified estimated time entered or not and verified by your project lead?
- Have you prepared a To-do list for your task execution plan and verified with your senior?
- If required, have you created a new branch in GIT?
- If task is related to API development

By following these steps, we can ensure that tasks are started efficiently, effectively, and with clear communication with the project team.

Before start work on API development Task

When backend developers start work on API development tasks, additionally they have to follow the below checklist.

- Have you defined the API endpoints and the type of API calls to be made?
- Have you discussed the API calling process and the authentication requirements?
- Have you communicated with the frontend/mobile development team that will be using the API?
- Have you discussed with the team the specific data they require in the API response and in what format?

While Working on a Task

- Have you added a progress comment on the task at the end of the day to update others on your work and any significant developments?
- Did you mention any difficulties or technical issues encountered during the task in a comment to keep the team informed?
- Have you tracked the total hours spent on the task, ensuring they do not exceed double the estimated time?
- If the total hours exceed the estimated time, have you discussed it with the project lead for guidance on managing the task effectively within the allocated time?
- This questionnaire addresses key aspects to consider when a task is in progress, such as communication, issue reporting, and managing time constraints effectively.

By following these practices, you can maintain clear communication, address challenges promptly, and ensure efficient progress on tasks during the development process.

After Task is Completed

To ensure better execution of tasks and effective reporting, developers and designers should consider the following checklist:

- Have you conducted comprehensive testing to ensure the task's functionality meets the intended requirements?
- Does the implemented feature align with the task description and cover all specified test cases?
- Have you optimized the code logic for improved efficiency and performance?
- Have you removed unnecessary junk or commented code from the final version?
- Are the naming conventions used in the code proper, consistent, and meaningful for easy understanding?
- Is the code well-commented and documented to facilitate comprehension by other developers?
- Did you adhere to the coding standards and guidelines specific to the technology and framework used?
- Is your code organized and structured in a way that promotes readability and maintainability?
- Does your code accurately implement the required functionality as specified in the task?
- Have you considered and tested edge cases to ensure proper handling of unexpected scenarios?
- Have you implemented appropriate error handling mechanisms to ensure smooth functioning of the code?
- Have you included sufficient error logs and debugging logs for efficient bug tracking?
- Have you pushed the code with the task ID and a clear commit message in all related commits?
- If you worked on a separate branch, have you created a merge request for the original branch after completing the task?
- Have you moved the next-day task to the "In Progress" list for a smooth workflow?
- Have you prepared accurate and detailed reports on task progress, including any challenges or delays encountered?
- Have you effectively communicated with relevant stakeholders to keep them informed of your progress and any obstacles?
- Has the completed task been moved to the "Code Review" list for a thorough evaluation by a team member or superior?

By following this checklist, developers and designers can ensure that their tasks are well-tested, properly implemented, optimized, documented, and adhere to coding standards. This promotes code quality, collaboration, and efficiency within the software development process.

After complete API development task

When backend developers complete the task of API development, additionally they have to follow the below checklist.

- Have you tested all possible API requests and responses using a dedicated API testing tool?
- Have you documented the API's expected requests and responses according to API documentation standards?
- Have you communicated with the frontend/mobile developers once the API development is complete?

Before Start Work on Research Tasks

Here are some additional tips for creating better content before starting work on technical R&D tasks:

- **Understand the problem:** Before you begin any research and development work, make sure you have a clear understanding of the problem you're trying to solve. This will help you stay focused and ensure that your efforts are aligned with the end goal.

- **Define success criteria:** It's important to define success criteria upfront so that you can measure your progress and determine when you've achieved your goal. This will help you stay on track and ensure that you're making progress towards your desired outcome.

- **Create a timeline:** Once you've defined your goal and success criteria, create a timeline that outlines the key milestones you need to hit in order to achieve your goal. This will help you stay organized and ensure that you're making progress towards your goal.

- **Identify risks and challenges:** As you're working on your research and development tasks, it's important to identify any potential risks and challenges that could impact your progress. This will help you proactively address these issues and keep your project on track.

- **Stay flexible:** Research and development tasks often require a lot of experimentation and iteration. It's important to stay flexible and be willing to adjust your approach as you learn more about the problem you're trying to solve.

Apply 5 Ws for Research.

To conduct effective research and development, it is important to apply the 5 W's technique. By following these tips, you can create better content and set yourself up for success in technical R&D tasks.

The 5Ws concept can be applied to ensure comprehensive planning and execution. Let's explore the 5Ws in this context:

- **What?:** The "What" in software development research and development tasks refers to the specific task or problem being addressed. It involves understanding the objective or goal of the task, such as developing a new feature, optimizing code, fixing a bug, or conducting performance testing.
- **Why?:** The "Why" in software development research and development tasks involves understanding the purpose and significance of the task. It includes identifying the underlying need or benefit that the task aims to fulfill, such as improving user experience, enhancing security, increasing efficiency, or aligning with industry standards.
- **Who?:** The "Who" in software development research and development tasks focuses on the individuals or teams responsible for executing the task. This includes identifying the developers, designers, testers, and other relevant stakeholders who will be involved in the task. Clear roles and responsibilities ensure effective collaboration and accountability.
- **Where?:** The "Where" in software development research and development tasks addresses the context or environment in which the task will be performed. It includes considerations such as the specific codebase, software module, or system component that the task relates to. Understanding the context helps in organizing and coordinating the development efforts.
- **When?:** The "When" in software development research and development tasks determines the timeline and schedule for executing the task. It involves setting deadlines, milestones, and priorities. Having clear timeframes helps in managing resources, coordinating dependencies, and ensuring timely completion of the task.

By considering the 5Ws in software development research and development tasks, teams can ensure a comprehensive understanding of the task, its purpose, the individuals involved, the relevant context, and the timeline for execution. This facilitates effective planning, collaboration, and successful completion of the task within the desired timeframe.

While Research Task is In-Progress

- Ensure that you attach any relevant reference links you come across during your research. These links can provide valuable insights and support your findings.
- Create a document and sample code to document your research findings and any experimental code you may have developed. This helps in sharing your research with the team and provides a reference for future work.
- Share your progress and findings with the team by communicating what you have learned or discovered during your research. This promotes knowledge sharing and keeps everyone informed about the progress being made.
- If task length has multiple days, you need to follow daily progressive comments as per general task guidline.

By following these practices, you contribute to a collaborative research environment, where valuable resources are shared, findings are documented, and knowledge is disseminated among team members.

While Work on Bug resolution

After a bug is reported and assigned to you, it is important to follow the checklist below:

Before Start Work on The Bug:

- Has the QA person provided the reference of the actual task or story related to the bug?
- Has the QA person written a proper description of the bug, explaining the issue clearly?
- Has the QA person provided detailed steps for reproducing the bug?
- Have you thoroughly understood the bug issue and its impact on the application?

After Resolving The Bug:

- Have you tested the affected feature thoroughly after fixing the bug to ensure it functions correctly?
- Have you updated the bug status in Jira as "resolved"?
- Have you conducted integration testing to verify that the bug fix has not caused any new issues?
- Have you tracked the time spent on fixing the bug within the same task? It is important to track time accurately.
- In the comments, have you mentioned below points which is very important for bug resolution
    - The root cause of the bug.
    - Bug resolution.
    - Preventive measures which will not happen in future.

Before Release in UAT/Production After Bug Resolve

Before releasing tasks to the UAT or Production environment, it is crucial to project lead ensure the following:

- Have developers reported bugs and issues that have been thoroughly resolved and tested after the bug fixes?
    - Verify that all reported bugs have been addressed and fixed properly.
    - Conduct comprehensive retesting to validate that the fixed features or functionalities are working correctly.
    - Ensure that the fixes have not introduced any new issues or regressions.
- Have developers conducted thorough retesting after resolving the bugs?
    - Retest the affected features or functionalities to ensure they are functioning as expected.

    - Validate that the bug fixes have resolved the reported issues completely.
    - Perform regression testing to verify that the fixes have not impacted other areas of the application.
- Have developers updated the bug tracking system or tool to reflect the resolution status of the reported bugs?
    - Update the bug tracking system with the appropriate status (e.g., "resolved," "closed," or similar) for each resolved bug.
    - Provide clear and concise information on the resolutions implemented for the bugs.
- Have developers made the necessary changes in technical documentation?
    - Document any changes made to the codebase or application as part of the bug resolutions.
    - Include any additional information or instructions that may be relevant for the production release.

By ensuring that all bugs and issues have been resolved, thoroughly retested, and properly documented, project lead can release tasks to the testing or production environment with confidence.

This helps maintain a stable and reliable application, provides transparency to stakeholders, and facilitates smooth operations in the live environment.

By following this checklist, project lead can ensure that bugs are addressed effectively, thoroughly tested after resolution, and properly documented to prevent their recurrence. This contributes to a more stable and reliable software application.

Checklist for QA

When QA is involved in testing, they should consider the following checklist for better execution of testing and bug reporting:

Before Start Testing on a Task

- Have you received the UAT release mail/notification, and is the task mentioned in the Task release list?
- Is the specific task listed in the "QA Review Pending" or "Development Completed" list?
- Have you thoroughly understood the requirements and description of the task?
- Does the task have proper descriptions and complete comments provided by the developer?
- For bug resolution tasks, has the developer mentioned the "Root Cause," "Resolution," and "Preventive Measures" in the comments?
- Have you prepared all the necessary test cases for testing this task?

- Do you have sufficient sets of data, inputs, and resources to test this task effectively?

After Testing Complete

- Have you tested all the test cases for the specific task and the associated story?
- Have you created a bug for a particular feature or functionality which is not done as per expectation or missing anything?
- Have you made a list of suggestions and discussed with the project lead for those suggestions?
    - If suggestions are very close to the task or story and need to implement it immediately then have you created suggestions on the same sprint?
    - For other suggestions, have you created a list of suggestions in the backlog?

When Create Bugs/Defects in Jira

When creating bugs in Jira, it is important to follow a comprehensive checklist to ensure accurate and efficient bug management. Here is an improved version of the checklist:

- Have you created a bug/defect ticket in the Jira and assigned it to the appropriate developer?
- Have you provided clear and reproducible steps for reproducing the bug in the issue?
- Have you indicated the severity and priority of the issue to prioritize its resolution?
- Is the bug related to a specific child task of the master task? If so, ensure it is properly linked.
- Does the bug stem from conflicts between multiple tasks being developed? Make sure to address this in the issue.
- Have you included the Jira task link in the issue description to establish a clear connection?
- Have you mentioned the issue link in the Jira task comment to provide visibility and context?
- Have you added a bug label to the Jira task to facilitate easy identification and filtering?
- Have you moved the Jira task to the appropriate category, such as the "Bug Found List" or a similar section?
- If no bugs are found, have you moved the task to the "QA Completed List" or a similar category to indicate its completion?
- Have you categorized bugs in any of the one or more from defined categories?

By diligently following this enhanced checklist, you can ensure that bugs are accurately documented, assigned to the right personnel, and effectively tracked within the Jira system. This streamlined bug management process contributes to improved efficiency and timely bug resolution.

After Completing Testing all Tasks/Stories:

- Have you sent a report/email to project lead and internal stakeholders with the number of issues found, if any?
- If no bugs are found and all tasks are working properly, have you confirmed that they are ready to go into production?

By following this checklist, QA teams can ensure thorough testing, accurate bug reporting, effective communication with developers, and a smooth workflow for moving tasks towards production readiness.