# Definitions

## Definition of Ready

A definition of ready (DoR) is used to determine whether work on a item is ready to be started. It is mandatory that any item has to fulfill DoR criteria by the time it’s moved to the Todo status.

Criteria to Qualify

1. An item can be nothing but a story or a production bug.
   1. Tasks has to be converted into a story.
   2. All bugs have to be reproducible in a production environment.
   3. All non-production bugs (eg:- Development, QA, Staging, etc.) has to be a part of a story that is WIP (Work In Progress).
2. Product owner must’ve signed-off the item. This should ideally be done in the backlog grooming session but its also possible to sign-off in any ceremonial/non-ceremonial meeting as well.
3. The item has to be estimated in story points. Estimation has to be finalized in the backlog grooming session.
4. The story point weight of the task has to be below 8 (Assuming Fibonacci estimation being used)
5. A development cycle for the story should be equal to or less than 5 days (excluding R&D)
6. A QA cycle for the story should be equal to or less than 3 days.
7. The task has to be clearly understood by the team (indicated by the story points).
   1. QA owners should have a clear idea of how to test the task. Test cases are not mandatory at this point.
   2. Developers should have a clear idea of how to develop the story.
8. The acceptance criteria have to be finalized and approved by the product owner the team has to have a clear idea of how to evaluate it during sprint review.
9. The story or bug has to have a business value signed-off by the PO.
10. All dependencies have to be identified and signed-off by the team.
    1. Development and QA leads has to sign-off the dependencies by development leads in a pre-planning session prior to backlog grooming session. The item should be signed off from the architecture lead (Either the Solutions Architect or Dev lead) and that should reflect in the relevant field in the item. The same goes with the QA lead sign-off.
11. There can’t be more than 3-point story point allocations per story for R & D. If it is more than 3, it should either be split into separate tasks or created as a separate story with clear goals and acceptance criteria.
    1. In such cases, BAs can act as QA to verify the success of the R&D.

**SUGGESTION**

1. NFRs must’ve been defined.



[Non Functional Requirements (NFRs)](https://inivos.atlassian.net/l/cp/1GXfUmMt)

What happens after.?

 When a story qualifies against the DoR, the task has to be moved to the “Todo” status from “Not Ready”.  The task is ready to be taken into a release planning and then to sprint planning.

 If there is a “Hold reason”, it has to be cleared.

What happens when a story doesn’t qualify against the DoR.?

If the story doesn’t qualify due to a blocker that isn’t possible to clear during the backlog grooming session, a “Hold reason” has to be set. For, tasks outside the sprint, “On hold” status shouldn’t be set.

The team should work toward clearing the DoR disqualification reasons.

## Dependancies

Following is a set of dependencies that has to be cleared before at different stages.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dependency** | **Item** | **When to Review** | **Respon sible** | **Reviewer** | **Done.?** |
| Requirements | User story | **BEFORE GROOMING** | BA | SM | Ok |
| BA diagrams |
| Acceptance criteria |
| UI | User interface - Figma | **BEFORE GROOMING** | BA | SM | Ok |
| Resources | Cloud resource provisioning & configuration (Dev environment) | **BEFORE SPRINT ST…** |  | Architect1 | Ok |
| Cloud resource provisioning & configuration (QA environment) |
| Devices |
| Software Licenses. |
| Subscriptions |
| Skills | Checklist all necessary skills required to complete a task. |  | EM | pending - Architect1 | Ok |
| Architecture | Architecture diagram (optional) | **BEFORE GROOMING** | EM | EM | Ok |
| Technical debt declaration. |
| Sign-off |
| Development | Database diagram | **BEFORE GROOMING** | **DEV TEAM** | SM | Ok |
| Sequence diagram (optional) |
| DevOps | Deployment diagram | **BEFORE GROOMING** |  | SM | Ok |
| Quality | Happy path test cases | **BEFORE PLANNING** | **QA TEAM** | SM | Ok |

Pending items to be added:

 External API specs - potentially under requirements or development.

**Dependency review Process**

A diagram of a project

Description automatically generated

**Dependency closure : Day 1 of sprint**

A diagram of a diagram

Description automatically generated

## Definition of Done

The “Definition of Done” (DoD) describes the quality standards for the Increment to be considered “done” and in a state that it can be effectively inspected. It creates transparency by providing everyone a shared understanding of what work was completed and what standards were met as part of the Increment. Without this crystal clarity, it’s not possible for stakeholders to provide informed and reliable feedback.

Criteria to qualify.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| All test cases are automated. Every merge to the QA branch runs these test cases. | **OPTIONAL** |  |
| Developer Test case runs. |  |  |
| QA owner test case runs. |  |  |
| Code review. |  |  |
| Test cases run with high quality production-like data. |  |  |

**Definition of Dev done**

 At least one cycle of test cycle done.  Added missing test cases.

 Code review completed and passed.

Definition of QA done.

At least one cycle of test done.

Add missing test cases from the previous run.

What happens when a story doesn’t qualify against the DoD?

If the story doesn't qualify against the definition of and QA Done , the story shouldn’t be presented at the Sprint Review.

Dev Done

## Story points

Story points are units of measure for expressing an estimate of the overall effort required to fully implement a product backlog item or any other piece of work. The best definition of story points is that they represent the **effort** to develop a user story or product backlog item. Effort is a question of time: how long it will take to finish something. Many factors go into determining effort, including

 The amount of work to do  The complexity of the work

 Any risk or uncertainty in doing the work

When estimating with story points, many things come into play: complexity, effort, risk, and volume. But ultimately, story points are an estimate of effort.

When estimating with story points, a point value will be assigned to each item. Some teams use a **modified Fibonacci sequence** (1, 2, 3, 5, 8, 13, 20, 40…); others use a **doubling sequence** (1, 2, 4, 8, 16…). The raw value assigned to story points is not important, what matters are the *relative values*. A user story that is assigned two story points should be twice as much effort as a one-point story. It should also be two-thirds the effort of a story that is estimated as three story points.

One of the main reasons story points are so valuable is that they allow team members with different skill levels to communicate about and agree on an estimate. Instead of arguing about how long it might take each team member personally to do something, teams instead can quickly say that this user story is about twice or three times as much effort as that user story. With story points, it’s all relative.

**Examples**

|  |  |
| --- | --- |
| **Story points** | **Example stories** |
| 1 | [TN-612: [CR] As a Transport Coordinator, I need to view all scheduled voyages, S](https://inivos.atlassian.net/browse/TN-612) [o that I can manage and coordinate scheduled voyages effectively](https://inivos.atlassian.net/browse/TN-612) [**DONE**](https://inivos.atlassian.net/browse/TN-612) |
| 3 | [TN-619: [CR] As a Driver, I need to see the pending voyage, So that I can plan an](https://inivos.atlassian.net/browse/TN-619) [d prepare for the voyage](https://inivos.atlassian.net/browse/TN-619) [**DONE**](https://inivos.atlassian.net/browse/TN-619)  [TN-616: [CR] As a Driver, I need to take a picture of the odometer reading, So tha](https://inivos.atlassian.net/browse/TN-616) [t I could send it to the Transport Coordinator](https://inivos.atlassian.net/browse/TN-616) [**DEV - TODO**](https://inivos.atlassian.net/browse/TN-616) |
| 5 | [TN-598: As an admin, I need to search for users, So that I can assign the searche](https://inivos.atlassian.net/browse/TN-598) [d user with a role (assigning the role isn't included)](https://inivos.atlassian.net/browse/TN-598) [**DONE**](https://inivos.atlassian.net/browse/TN-598)  [TN-499: As a user, I need to log into the TMS system, So that I can access the sy](https://inivos.atlassian.net/browse/TN-499) [stem and its features/functions.](https://inivos.atlassian.net/browse/TN-499) [**DONE**](https://inivos.atlassian.net/browse/TN-499)  [TN-53: As a Driver, I want to view all the completed voyages, so that I can review](https://inivos.atlassian.net/browse/TN-53) [the details of the past voyages](https://inivos.atlassian.net/browse/TN-53) [**DEV - TODO**](https://inivos.atlassian.net/browse/TN-53) |
| 8 | [TN-718: As a Manager, I need to view trip requests in a way that distinguishes bet](https://inivos.atlassian.net/browse/TN-718) [ween my own trip requests and those of my subordinates](https://inivos.atlassian.net/browse/TN-718) [**DEV - TODO**](https://inivos.atlassian.net/browse/TN-718)  [TN-602: As a Driver, I need to validate my email using both Azure authenticator](https://inivos.atlassian.net/browse/TN-602) [(MS authentication app) and system validator, So that I can log into the Mobile app](https://inivos.atlassian.net/browse/TN-602)  [**DEVELOPMENT IN PROGRESS**](https://inivos.atlassian.net/browse/TN-602) |
| 13 | [TN-591: As a Transport Coordinator, I need to receive an automated email, So tha](https://inivos.atlassian.net/browse/TN-591) [t I know that the voyage has started](https://inivos.atlassian.net/browse/TN-591) [**DONE**](https://inivos.atlassian.net/browse/TN-591) |

Learn from past estimates.

Retrospectives are a time for the team to incorporate insights from past iterations–including the accuracy of their estimates. Many agile tools track story points, which makes reflecting on, and re-calibrating estimates a lot easier. Try, for example, pulling up the last 5 user stories the team delivered with the story point value 8. Discuss whether each of those work items had a similar level of effort. If not, discuss why. Use that insight in future estimation discussions.

Remember the Definition of Done

A story point estimate must include everything involved in getting a product backlog item all the way to done. If a team’s [definition of done](https://www.mountaingoatsoftware.com/blog/clarifying-the-relationship-between-definition-of-done-and-conditions-of-sa) includes creating automated tests to validate the story (and that would be a good idea), the effort to create those tests should be included in the story point estimate.

**Further readings:**



[What Are Story Points and Why Do We Use Them?](https://www.mountaingoatsoftware.com/blog/what-are-story-points)

[What are story points in Agile and how do you estimate them?](https://www.atlassian.com/agile/project-management/estimation#%3A~%3Atext%3DStory%20points%20are%20units%20of%2Cwork%2C%20and%20risk%20or%20uncertainty)

## 10 Types of Technical debts

Technical debt is a concept that plays a significant role in software development. It represents the compromises made in the software's design, code quality, or architecture to meet short-term goals or deadlines. While taking on some technical debt is often necessary, accumulating too much can lead to long-term problems. In this article, we'll explore the various types of technical debt that could exist in a software project, helping you identify and manage them effectively.

Code Debt:

This is perhaps the most common form of technical debt. Code debt includes poorly structured or undocumented code, inconsistent naming conventions, and shortcuts taken to meet tight deadlines. Over time, code debt can make the software difficult to maintain and extend.

Design / Architecture Debt

Design debt refers to compromises made in the software's architecture or overall design. It might involve taking shortcuts to expedite development without considering long-term consequences. This can lead to rigid or inflexible architecture that's challenging to modify as requirements evolve.

Testing Debt

Testing is a critical aspect of software development. Testing debt occurs when there's insufficient or inadequate testing due to time constraints or a lack of proper test cases. This can result in undetected bugs and issues that become more challenging to address as the project progresses.

Documentation Debt

Incomplete or outdated documentation is a common form of technical debt. It can hinder new team members from understanding the software's functionality and structure. Documentation debt often arises from neglecting to update documentation when changes are made to the software.

Dependency Debt

Many software projects rely on third-party libraries, frameworks, or components. Dependency debt arises when these external dependencies are not properly managed. This can lead to compatibility issues, security vulnerabilities, and difficulties when upgrading or replacing components.

Performance Debt

Performance debt occurs when software performance optimization is postponed. This could involve inefficient algorithms, resource- intensive operations, or slow database queries. Accumulated performance debt can result in a sluggish application that frustrates users.

Security Debt

Neglecting security considerations can lead to security debt. This includes not applying the latest security patches, not following secure coding practices, or failing to conduct thorough security audits. Security debt can expose the software to vulnerabilities and pose a significant risk.

Infrastructure Debt

Infrastructure debt pertains to the underlying infrastructure and tools used in the software project. Outdated server configurations, unsupported software versions, or suboptimal hosting choices can result in increased operational issues and maintenance overhead.

Process Debt

Process debt is related to the development process itself. It includes issues like a lack of automated testing, inconsistent code reviews, or poor communication within the team. These process-related challenges can impede productivity and quality.

Tooling Debt

The tools and development environments used in a project can accumulate technical debt if they are outdated or not effectively integrated. Outdated development tools, inefficient build pipelines, and unsupported IDEs can hinder development speed and efficiency.

Recognizing these various types of technical debt is essential for managing and mitigating their impact. While some technical debt is inevitable, it's crucial to strike a balance between short-term expediency and long-term sustainability. Regularly assessing your software

project for technical debt and addressing it as part of ongoing development efforts is key to maintaining a healthy and efficient software ecosystem. By managing technical debt proactively, you can ensure your software remains adaptable, maintainable, and secure in the long run.