Scrum Execution

Daily

1. In the Scrum Daily every developer gives a short overview of

- *What has been achieved on the last working day?*
- *What is planned for today?*
- *What are current issues they are facing (dependencies, blockers, capacity/estimation …)?*

2. **Jira Tickets** should always be the **Single-Point-of-Truth**, so keep it updated/documented with everything relating to that ticket
3. Issues, questions and dependencies regarding User Stories should **always be tracked as comments** in the respective Jira tickets as well
4. Everyone should therefore **enable notifications** on their Jira issues

At the end of each Daily, the QA/Tester should address the following points:

- Open questions and clarifications regarding User Stories in UAT.
- New bug reports either identified by the QA Tester themselves or through <kanban board or google sheet>
    - If the team agrees that a bug report is indeed a bug, the QA Tester will create the respective bug in Jira.
    - The team will then assign one of the developers (typically the one with the most knowledge in that particular topic) for the resolution of the bug.
    - Depending on the priority of the bug, it will be planned into the current or upcoming Sprints.

Keep track of the percentage of Stories that have been pushed to UAT per developer.

Idea Backlog

A place for gathering **very high-level ideas** (purely conceptual) **about the future of the product**. Anybody feel free to add your ideas here <google doc or confluence page>

Product Backlog (Epic Drafts)

The product owner should **create drafts for Epics** (e.g. a feature that he wants to have in the product - "Commenting") and **then prioritize** it:

- **Copy** the <epic draft template> over to the section <epic management confluence page>
- Then **rename** the title of the page to reflect the desired feature.
- **Fill out** the required information as well as possible.
- **Prioritize** Epic Draft by reordering it in the section mentioned above.

The Epic Drafts are meant **for initial planning** of an Epic **only**! Once an Epic Draft has been translated to actual User Stories in the Jira Backlog (see below), the respective Epic Draft document in Confluence **becomes obsolete** and does not have to be kept in sync!

Obviously, not every User Story is 100% predictable, foreseeable or Epic-related at all. For these kinds of User Stories refer to “Creating Stories” below.

Creating Stories (Refinement)

Most User Stories are **drafted in the Refinement** meetings with the entire team. Here, we go through the **Epic Drafts** (highest priority first) and **decompose them down into drafts of individual User Stories** in the respective template that was created by the Product Owner (see above). This includes specifying the title, description and acceptance criteria (maybe even rough estimates) for each User Story draft. It is also important to **identify dependencies** between User Stories.

Once all stories are drafted for a given Epic Draft, both the Epic and User Stories are created and **put into the Jira Backlog** (Epic manifestation).

Other User Stories (Post-manifestation and Non-Epic-related)

Obviously, there can be instances when

- **additional** User Stories are **necessary for the completion of an Epic** after its manifestation, i.e. its User Stories have already been created (e.g. follow-ups, unforeseen requirements or dependencies) *or*-
- **non-epic related** User Stories (e.g. small features etc.) need to be created

These kinds of User Stories can be **suggested and drafted** in the <user stories draft confluence page>document. User Stories in this document will also be **discussed in the Refinement** meeting. Analogous to our Epic Drafts, when a draft for a User Story is completed and confirmed it can be created in the Jira Backlog (User Story manifestation) for the Sprint Planning.

For changes to existing User Stories refer to “Changing User Stories” on the bottom of this document.

External Feature & Change Requests

External Feature & Change Requests can be brought to the developing team’s attention through the dedicated page <external feature draft confluence page>

In every Refinement Meeting, the team will go through the new entries from that list and discuss it according to the process outlined in the Confluence Page.

Title Naming Conventions

The **title** of a User Story should **always be expressive** of its shippable goal. Additionally, we utilize several **prefixes** to further differentiate individual User Stories:

Aa Prefix	Meaning/Output
Untitled	
Design	Mockups and design specifications incl. assets (Invision, Abstract)
Style	Design is styled as a component (either as HTML/CSS only -or- Angular component)
Frontend	Implementation of logical components in the frontend.
Backend	Implementation of logical components in the backend.

- The prefix is separated by a "|" symbol, e.g. "Design | Button for printing the current page".
- Prefixes can also be **combined**, e.g. "Design + Style" or "Frontend + Backend".
- Additional prefixes can be used to further **describe the component** that is involved, e.g. "Backend | Data Provider Service | Create an endpoint for returning basic customer data".

If a User Story **cannot be placed** into any of the above categories it will simply be created **without a prefix**.

Estimation Meetings

Estimation meetings should be held mainly by the development team (but anyone is welcome to join) **if there's still User Stories in-estimated** or if re-estimations are needed (both in Jira and in the previous Epic drafts).

If re-estimations turn out to be necessary, the case should be argued with the Product Owner in the Daily and in the comments of an individual User Story (for documentation purposes).

Estimating Story Points

Estimations are made in units of Story Points, 1 Story Point (SP) *roughly* equals 2 hours worth of work. We utilize the **Fibonacci sequence** for applying Story Point estimations to individual User Stories:

Story Points	1	2	3	5	8

User Stories **bigger than 8 SPs** should always be **split** into smaller User Stories. For User Stories **with 8 SPs** splitting should at least be considered.

According to Scrum, there should be no correlation between Story Points and actual working hours. However, given the billing model being used this correlation is inevitable here.

Overhead Markup

Estimation should always **include time buffer (markup)** for

- Regular & Ad-hoc Meetings (markup calculation example below)
- Writing Tests (Unit + Integration)

- Documentation (Confluence, Code, ...)
- Internal Merge / Code Reviews
- Resolving Merge Conflicts
- Deployments to Acceptance (Release Branch)
- Pipeline Issue Resolution (communication)
- Preparation for UAT (Screenshot, Videos, Test Data & Instructions, ...)
- UAT communication + eventual corrections
- Preparations for Refinement & Estimations (Draft Ownership)
- Bug Fixes

It is recommended to **track the actual workload** (in hours, incl. overhead) so that each developer can assess how good the estimations in terms of Story Points were (accuracy and velocity).

Calculating the Regular Meeting Overhead:

Meeting	**Duration**	**Frequency Per Sprint**	**Duration**
Pre-daily	00:10	8	01:20
Daily	00:15	8	02:00
Refinement	01:00	2	02:00
Estimation	01:00	2	02:00
Review, Retro and Planning	02:00	1	02:00
Total			**09:20**

(based on 2 weeks sprint)

(Design Weekly and ad-hoc Meetings not included in the calculation)

Do not estimate User Stories **too low or too narrow** as this inevitably causes the **quality to suffer**!

Design Weekly

As design-related User Stories constitute a special case in our Scrum execution, an independent meeting is held once a week (if necessary) in order to

- discuss the current design-related User Stories (feedback, iterations, blockers, …) and make decisions moving forward.
- drafting design-related User Stories for the upcoming Sprints.

The Design Weekly is **only to discuss the current open questions** and make decisions on the details of the designs - it is not intended to be a deadline or a design-specific Review meeting!

There can be situations where these **User Stories have an unclear scope** or need a certain amount of **feedback loops and iterations** (e.g. “Design a logo for the project”). Hence, some User Stories will be **estimated** not based on their scope but on the **amount of effort (i.e. iterations) that should be put into it**. We mark these kind of User Stories with another Prefix “Iteration” (as a component, e.g. “Design | Iteration | Create a logo for <Project name>”).

Sprint Planning

The Sprint Planning **marks the beginning of a Sprint**.

In the meeting, the Product Owner should state his “Business Objective” and then - together with the team - try to draft a “Sprint Goal” from this. For this, the Product Owner (with the team) will determine **which User Stories** (from Jira Backlog) **to put into the Sprint Backlog** of the next Sprint (depending on the total amount of Story Points, dependencies etc.).

The selected User Stories will then be **prioritized**.

If necessary, it is possible to include a little Refinement session into the Sprint Planning.

Assigning User Stories

Assignments of User Stories to individual developers are **best done ad-hoc during the Sprint** execution. Thus, each developer simply picks the next story and no developer is overloaded.

As a general rule, the User Stories with the **highest priority should always be done first**!

That being said though, for us there are some natural constraints (mainly scope of the individual developers and capacity planning) that make it necessary to **already have a rough plan** which developer is going to process which User Stories **during the Sprint Planning**.

Review

The Sprint Review (together with the Retrospective) **marks the very end of a Sprint**.

All User Stories **should be done and tested** by the time of the Sprint Review. Everything else *should be considered* moving into the next Sprint (i.e. only include shippable deliverables).

After a successful Sprint Review the Sprint (incl. all its User Stories) should be closed by the Product Owner.

Retrospective

Feedback about the Sprint by the entire team and what to try to do better next time (focus on collaboration and the process itself).

Retrospectives should be held with funretro.io and **archived** in the Retrospective Log document (also refer here for the structure of the boards and the Retrospective in general).

Immediate action items identified in a Retrospective (or elsewhere) can be placed in the next Sprint as **Tasks** while mid- and long-term action items can be kept in a continuous log.

Impediments / Changing User Stories

Impediments of any form (information missing, blockers/dependencies, …) should **always directly be made aware of!** This includes changes that might need to be made for an existing User Story like e.g.

- **Moving** a User Story out of or into the current Sprint,
- **Re-estimation** of an existing User Story of the current Sprint,
- **Changing the scope** (description, acceptance criteria, …) of a User Story.

It is important to note that these **changes should never be simply applied** by yourself. It is required to always **discuss** the requested changes with the entire team:

1. Mark the impeded User Story with a **flag** 🚩 to indicate a required action.
2. **Add a comment** to why the individual User Story is impeded (e.g. unforeseen dependency, blocked by decision etc.)
    a. **Tag** whoever is needed to clarify the issue (incl. the Product Owner and Team Lead).
    b. Additionally, make everyone aware in the **next Scrum meeting** (next Daily the latest!).
3. **Resolve the issue** and remove the flag.

Flags should **only be removed** once the issue was **confirmed** to be resolved.

In general, **changes** to existing User Stories **should be avoided** if possible.

Removing obsolete User Stories

It may happen that manifested User Stories become obsolete and are not needed anymore. In that case they **should not be deleted** but still be kept for future reference.

This must only be done by the Product Owner.

1. Mark User Story with **summary prefix** * OBSOLETE *.
2. **Comment the reason** why this User Story was deemed obsolete.
3. **Unassign** the User Story and **unset any estimations** put on it.
4. **Close** the User Story.

Weekly Meetings

Sprints start in **round** weeks (<year>). Meetings are typically held on <slack, skype, google chat or whatever>.

Name	Rhythm	Time
(Pre-Daily)	Mo. – Fr.	09:05 – 09:15
Daily	Mo. – Fr. (except Review)	09:15 – 09:30
Refinement	Th. (first week) Tu. (second week)	09:30 – 10:30 09:30 – 11:00
Estimation	Th. (second week)	09:15 – 10:45
Design Weekly	Mo. (weekly)	13:30 – 14:15
Review, Retrospective & Planning	Th. (biweekly, end of sprint)	12:00 – 14:30

Participation

In order to keep the participation high, it is strongly encouraged to turn on the cameras during all meetings.

Absence from Scrum meetings

In general, if **someone is missing from the Scrum meetings** it is expected that either

- a **written update** with the necessary information is given. *-or-*
- all necessary information are available to any of the **other present participants** *and-*
- all **open questions and current blockers** have been made clear through **flags and comments** in the respective User Stories.

Sprint Capacity

The actual capacity per Sprint and Developer is tracked in Story Points per Sprint.

Developer	Capacity	Scope
Prayas Poudel	20	Development
Ayush Bajracharya	16	Design
Manish BC	20	Design, Style
Pradipta Bista	20	Fullstack
Slesha Tuladhar	18	Fullstack

Developer	Capacity	Scope
Total	94	Untitled

Exceptions

- **Vacation / Illness:** Capacities will still be met by substitute resources.
- **Holidays:** Local holidays will be excluded from the Sprint's total amount of Story Points.

Billing Model

Currently, <company> bills per Story Point which has some implications on how Scrum is executed and lived not only by the development team but also for project management in general. The following pages describe how Scrum is applied for <project> giving the current billing model.

Idea Proposal

In order to still be able to make use of **all of Scrum's core features and advantages**, we'd have to find around the billing model impacting the process since it is the contract between <party> and <company> that imposes these implications and therefore should be treated as immutable.

One idea would be to decouple billing and the development process by differentiating **"billing" or "capacity" Story Points** and **"development" Story Points**.

"Billing" or "capacity" Story Points would represent the actual **total capacity** that the developers have per Sprint (e.g. in our case 110 Story Points – with no public holidays, extra resources etc.). This is always the amount of work that has been put into the Sprint by the developers and equals the Story Points that would be billed per Sprint by <company>. These Story Points **don't have any other purpose** and will not be used in any part of the development process and are thus entirely decoupled.

"Development" Story Points on the other hand can then be **used for their original purpose** and will be utilized to execute the Scrum process to its full potential, e.g.

- defining User Stories as they are meant to be (not splitting Frontend, Backend, Design etc.),

- better team work as User Stories are always a team effort (requiring most of the team to work on a single User Story),
- Story Points indicate the complexity of User Stories (not the actual effort),
- less pressure on Estimation, Sprint Planning meetings etc.,
- less pressure to exactly match workload and estimations,
- more flexibility for the whole process in general.
- …

The “Development” Story Points **don’t have any relation** to the “Capacity” Story Points and thus **are irrelevant for billing** entirely!

To be discussed: Overtime, etc.!

Story & Development Workflow
User Story Workflow

State “To Do”

Developer picks a User Story to work on (depending on their priority) either from the list of yet unassigned User Stories or from the User Stories that have already been assigned to himself/herself.

State “In Progress”
Developer implements the User Story - see section "Development Workflow" for a detailed explanation of this step.

State “(Internal) Review”
Development work on the story was finished and a merge request was created. The merge request will be reviewed by (and discussed with) another developer and then merged to the development branch (see “Development Workflow”).

State “Testing”
After a merge request was confirmed and closed, the User Story is reflected on the development stage – for making it ready for testing by an external tester, it should additionally be pushed to the acceptance stage (using our release branches) like so:

- Pull the branch develop
- Checkout and pull the current release branch. It has the form release/0.<sprint number>.0.
- Run git merge --no-ff develop to merge the current state of the develop branch into the release branch.
- Confirm the commit message. Per default the “Vim” editor will open. You can confirm the commit message there by typing :wq and hitting enter
- Push the release branch.

All required documentation for successful testing (e.g. Postman collections, walkthrough descriptions, videos, etc.) should be added to the User Story as a comment.
We decided not to assign the User Stories to the tester anymore as this causes a lot of confusion. Same goes assigning it to the Product Owner when a User Story is done. Thus, the User Story always stays assigned to the developer that integrated it.

Instead, the tester should be tagged along with the documentation for testing as a comment of the individual User Story. In addition to that, he should always check for new testable User Stories using the Sprint Board himself.

Scope of Testing

- In general, all User Stories should be officially tested.
- Design Stories are reviewed by the Product Owner himself either during the weekly Design meetings or in the Sprint Review.

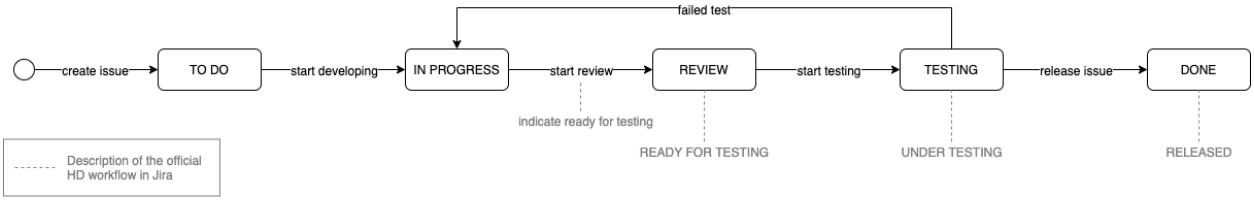
User Stories should be made available for testing as soon as possible from the beginning of the Sprint (and not just towards the end of it)!

State “Done”

Story was successfully tested and can be closed by the Product Owner.

Development Workflow

To be discussed and filled with the entire development team. <team lead>


failed test
create issue
TO DO
start developing
IN PROGRESS
start review
REVIEW
start testing
TESTING
release issue
DONE
indicate ready for testing
READY FOR TESTING
UNDER TESTING
RELEASED
Description of the official HD workflow in Jira