Writing User Stories

Why we have this Process

What is a User Story

User stories are a Scrum method for writing functional **requirements**. They define the functionality of a software system from the **user perspective**. The standard user story format is:

As a < type of user >, I want < the system to do this> so that < some outcome / reason >

In a Scrum environment, they replace requirements documents. User stories are **functional requirements**, meaning that they define **what** a system needs to do, but **not how** the technical solution should look like. User Stories are also the central **unit of work** in Scrum. To the Product Manager and outside world **progress** is defined in terms of completed user stories, and work is **estimated** no lower than user story level. Some teams prefer to split up user stories into tasks (called subtasks in Jira), i.e. to split work over multiple developers. This is however not done or tracked by the Product Manager. User stories can be in one of two places: The **Sprint Backlog**, when software development has started, or the **Product Backlog** for when development has not yet started.

Why User stories are important for Product Success

User stories act as **a reminder of what the team has agreed upon to develop** during the Backlog grooming sessions. Without user stories, the developers **wouldn't know exactly what to develop** and the testers wouldn't know what to test. The format also ensures product success by ensuring the requirements always address a **clear user need**. User stories are small and clear by there nature ensuring everybody has the **same understanding** of what is needed. **Additionally**, the user story format ensures the **reason why** something is developed is always clear which prevents misunderstandings.

Context and Scope

- User stories are written in the **Design process** as part of **Backlog grooming** by the **Product Manager**.
- User stories provide the Product Manager and **stakeholders** insight in the "tasks" the team is working on and related requirements.
- User stories are stored in **Jira** in the **Product Backlog**.
- User stories are refined (detailed, discussed, etc) in **Backlog Grooming** sessions and when they reach the top of **Product Backlog** they might be put into the **Sprint backlog** during a **Sprint Planning meeting**.

When should the process be performed?

Writing user stories generally starts **from the beginning of the Design** process

Writing and improving user stories is an **ongoing process** that does not stop until software development has also completed, as even during the Delivery user stories might need to be clarified or split.

Typically the Product Manager has **several weekly meetings** with either stakeholders or the team to gather input or feedback on the stories (on the top of) his Product Backlog.

Who performs the process

- Product Manager
- Reviewing is done by peers (other Products Managers), Stakeholders and the scrum team.

Process steps - Theme, Epic, User Story and Requirements Writing

All steps below are the responsibility of the Product Manager, but you want to work in parallel, and together with UX&UI Designer(s) so that i.e. user stories and UX stays aligned and is consistent at the end of the process.

1. Start **high-level requirements analysis**. The goal is to learn and get an overview, not to write a requirements document. However you can document high level requirements you encounter in the section "High level requirements" on the Extended User story mapping template (TODO: link to this page when added). Also for point d. Competitor Analysis you should document what you found, in the confluence section "Feature Discussions" on the page related to this theme (TODO: add a section for links to relevant feature discussions to the user story mapping template)
    a. **Top-down requirements analysis** (coming from above you in a hierarchy, long-term vision and goals, or laws, etc). Gather, discuss and study:
        i. Relevant company goals or OKR's
        ii. Product Vision, Product KPI's, Persona.
        iii. Roadmap **Theme** and Requirements already gathered during Roadmap theme preparation.
        iv. Theme Business case
        v. Requirements from relevant stakeholders
        vi. New external API's to be used
    b. **Bottom-up requirements analysis** (requirements coming from customers, current processes, and systems). Gather, discuss and study:
        i. Relevant existing screens, current system architecture, API's, manual processes/workflows
        ii. Relevant data: analytics/revenues/costs / other data points (i.e. for a fraud reduction feature current fraud costs/types /patterns. For a new service: how often was it requested or how many times is a currently a workaround solution used).
    c. **User Need analysis**
        i. Contact Customer Support for input relevant to the theme (support has daily contact with customers. Not talking to them is unacceptable).
        ii. Study relevant User Interview (summaries) and User Test outcomes (from Discovery process)
        iii. Describe the overall Job to be done (JTBD) or Outcomes the User is trying to achieve.
        iv. Describe the **Persona's**.
    d. **Competitor analysis**
        i. Look at least 2 competitors, study their service, put screenshots of their relevant service in the project folder.
2. **Start working on the Design** (Specifications and UI).
    a. Start writing **Epics** to document the requirements in Jira. Only titles are mandatory. Follow the User Story title naming convention below. Example epics "Order eBook", "Download eBook". Each user role type will have a separate Epic (i.e. buyers, sellers, admins).
    b. Map out the **User Journey for the whole theme (per Actor)**, using Steps. Use the **Extended User Story map** for this Some UX designers like to create a **Customer Journey Map** in parallel which is also based on these steps.
    c. Per step start writing relevant **User stories**. Read the paragraphs below about correct user story titles, how to split them up and what makes good user stories.
    d. Initially, start with just the **User Story title**, but add them directly to **Jira** as issue type **Design User Story**. (If you want to separately track the status of the UI Design part of User Story in Jira you can create subtasks for that, however, you can also use Invision for that).
    e. Prioritize the User stories together with the UI&UI Designers in Jira in the Design Backlog. Again the **User Story map** is a great tool to help set priorities.
    f. Start **working on Designs** in the Design Backlog priority order. This includes adding user story description including **Acceptance criteria** (please review the linked page!) and UX&UI Designs.
    g. In parallel work over **overall requirements artifacts** as soon as needed for a user story, and only the part that is needed.
        i. the **Business Object model** to show relationships between entities and **Entity Definitions** to name and precisely define all entities in the system (i.e. "Order", "NewsItem")
        ii. **State diagram** in case of cross-role workflows or entities for which correct state (transitions) are crucial (i.e. Order, Project)
        iii. **Workflow diagrams** using UML Activity Diagram for complex human or automated workflows (an entity i.e. a task goes through several stages over several days)
        iv. UX might want to start on the **Styleguide** and set up a **Design system** (reusable components) and start on the **Information Architecture**.
    h. If any **issues** come up during design that somehow prevents moving on what a certain part of the design (i.e. a decision has to be made by a stakeholder, a lawyer has to check something, some API is broken):
        i. Do your best possible assumption and continue the design using that.

        ii. Document the issue on an **Issue list** in Trello. Per issue be very clear about the problem and impact, the proposed next step, the owner of the next step of the priority of this issue in relation to other issues.
        iii. Contact the relevant person about the issue.
3. **Regularly** gather **feedback** on the Design while it is in progress (again together with UX&UI Designer(s)). The **User Story map** provides a great tool for this meeting as it gives a process overview.
    a. **Architects** about the technical feasibility. Ask the CTO or architect to start working on the technical architecture, tool, and framework selection, etc..
    b. **Peers** for peer reviews (i.e. other PM's, Head of Product). Recommended: weekly.
    c. **Stakeholders** (i.e. CEO, Head of Customer Support)
    d. From end users using **User Tests** on a (partly) clickable Design in Invision. Recommended: bi-weekly.
4. Gather relevant **Scrum team** for **Questions, Feedback, and Estimation** in a Backlog grooming session.

Quality criteria: What makes a good User story

Parts of a User story

- **Title** - I.e. View Authors.
- **Description** containing:
    - **Full User story** - i.e. As a Blog Administrator I want to see all Authors so I know who has access to Blog Writing functionality
    - **Link to clickable Design -** i.e. https://projects.invisionapp.com/d/main#/projects/prototypes/12345
    - A table with **texts that are not visible**, i.e. error messages in the primary language of the app (Responsibility of the UXUI Designer)
    - **Other** information that will help both stakeholders and the team understand the user need.
    - **Acceptance criteria** (See: Writing Acceptance Criteria). Limit yourself to circa 1-3 acceptance criteria the **cover the most important outcomes** of the story (not the function, i.e. not which fields should be mandatory, default values, etc)
    - **Detailed requirements**. If needed, detailed requirements.
        - Do **not** include things that can be seen in the UI Design (i.e. which fields/buttons are present). You *can* include details like which fields are mandatory (if not clear from i.e. data definition table) or what their default values are. An example requirement is: The email address should be validated against ISO-1234.

User Story Title Naming

Consistency in how we **name** our user stories makes it easier for everybody to understand the stories.

Good Examples

- Login
- Set new Password
- View Sold Car
- Add User
- View User
- Edit User
- Delete User
- View Users
- Filter Search results
- Filter Search results by radius around a coordinate
- Send Daily Update Email
- Select Appointment Time slot
- Review and approve Contract

Bad Examples

- Filter Search **R**esults **B**y **R**adius **A**round **C**oordinate - wrong, see rule 1
- **Create** User - wrong, see rule 2
- View **list** of Users - wrong, see rule 3
- Select Appointment timeslot **at Doctor for Itch** - wrong, see rule 4

The generic rules

1. Do **not capitalize all words**, only verbs, entities and important Entity **statuses** can be capitalized too.
2. Use the default naming for the **CRUD** list (Add, View, Edit, Delete)
3. Use the **plural form,** i.e. 'Members' to indicate **multiple** items can be shown, but not "View Members", not "View *list* of Members".
4. **Leave off** things that are already clear from the Epic or theme **context,** i.e. Reset *User* Password *in IOS in Foobar app*.
5. You can include the **Actor** in the title if there are **multiple actors** (i.e. Dealer and Player), but the verb stays in **singular** form, so Dealer Start Round, not Starts Round. The reason for this is that in our minds we are saying "As a Dealer, I want to Start a Round so...".

Splitting up user stories

Good Examples

Search Orders incl autocomplete	Well sized.
View (Orders) Search results	Well sized.
Filter Search results	Well sized.
Filter Search results by radius around coordinate	Stories can be always split up if they otherwise would be too big (i.e. more than 3 days work)
Upgrade firewall to 2.0	Important Technical tasks that don't belong to other stories may be added if they belong to the team.
Send Daily Update Summary Email	System initiated actions and background procedures like scheduled tasks are stories too.

Bad Examples

Book Hotel Room	Too big. Several screens touching many Entities.
Register and fill Profile	Too big. If a story deals with two quite separate entities (Account and Profile), create 2 stories,
Register - step 1 - enter Email	Too small. “Register - Step 1 - Setup Account” would be fine if registration consists out of multiple steps.
Sort Search results	Too small. Although it might require backend call, it doesn't change things.

General Guidelines

While reading these examples you might have distilled some patterns. These are some general guidelines:

- **Stories can be always split up** if they otherwise would be too big (i.e. **more than 3 days work**)
- As a rule of thumb for splitting up stories: If a story **updates multiple entities** (i.e. Person and Address), **split it** up (i.e. Edit Contact, Add Address). For viewing you can combine multiple i.e. View Contact and Addresses). **Actions** that go further than just saving fields justify a story too., i.e. Reset Password
- Scheduled or System-initiated actions like **scheduled tasks** are stories too. (As a newsletter subscriber I want to receive the daily status mail every day at my 13:00)
- **Never user System as an Actor** (in the long user story format), always translate it back to the user you're doing it for, i.e. the newsletter subscriber, see example above))
    - Use the **WHY's approach**, why does the user really want this i.e. "As the backup operator I want system backups to run every day at midnight have the backup result stored in the backup log" But why does he want this? "As a backup operator, I want them to be able to restore the database from backup so that..."
- Don't write **negative user stories** about what's **not allowed**, but write the positive opposite, i.e. Not: As an admin, I want users with an expired license being blocked from login but: As an authorized user with an active license, I can login to the system.
- Important **Technical tasks that don't belong to other stories** may be added too as stories if they belong to the team.

Focus on the outcome, not the solution at first

A typical User story beginner mistake is going into too much (technical) detail, i.e. "As a registered User I want to click on the reset password link on the homepage so I can ... ". By writing a user story in this way you don't allow the UX designer and other team members to come up with the optimal solution. **It is the role of the product manager to provide context, situation, motivation and expected outcomes, basically to take the user perspective and zoom out.** Further detailing should only follow based on team discussions that at least involve the UX designer. This might lead the team to decide on a solution without a password (yes they exist) or at least to ensure that the user doesn't have to enter his email again on the forgotten password screen. However, the **user stories do need to be detailed at some point**, i.e. by splitting them up into smaller pieces, adding acceptance criteria, defining business rules or flows, etc. For most teams, this point is when a clickable prototype is completed. Teams that actually do user testing on their clickable prototype before starting development can wait until the testing is done (lean means being done "just in time").

Job Stories

A recent trend is defining "Jobs to be Done" when analyzing requirements. This in line with the importance of **providing context, situation, motivation and expected outcomes** when writing requirements. A JTBD suggestion is to replace Epics or even User stories with "job stories" following the format: When [Situation] I want to [Action] So I can [Expected Outcome] (also known as: **Given** SITUATION **When** ACTION **Then** SYSTEMACTION The format itself is not very practical for scrum: It effectively means writing a story for every situation-outcome pair which would lead to a too small unit of work. However clearly defining the situation(s) and the outcome(s) from the user perspective should be part of every user story. I.e. a bad story is "I want to add a Task so the task is stored in the system". This is not an outcome with real **user value** nor recognize the context. A better one would be "...so I can easily decide what task to work on next without becoming overwhelmed".

Further guidance on defining and splitting up stories

INVEST is an acronym which encompasses the following concepts which make up a good user story:

- Independent
- Negotiable
- Valuable
- Estimable
- Small
- Testable

Let's cover each of them with a simple explanation.

- **Independent**: Stories should be as independent as possible. Why is this important? It allows for true prioritization of each and every story. For a restaurant app the stories "View menu items for a category" and "View menu item categories (i.e. starters)" should be combined into one story as the user wouldn't be able to view menu items without the categories, thus the story is not independent.
- **Negotiable**: It can be changed in discussion with the team and stakeholders during backlog grooming (until it's put into a sprint).
- **Valuable**: If a story does not have discernible value it should not be done. Period. Hopefully, user stories are being prioritized in the backlog according to business value, so this should be obvious. Some people say each story should be valuable to the customer or user. I don't like that way of thinking because business value encompasses more than just customer or user-facing value. It includes the internal value which is useful for things which are normally called "non-functional requirements" or something similar. I prefer to say the story has value to the "user" in the user story. In this way, it is clear who is to be satisfied. Finally, remember the "so that <value>" clause of the user story. It is there for a reason – it is the exact value we are trying to deliver by completing the story!
- **Estimable**: A story has to be able to be estimated or sized so it can be properly prioritized. A value with high value but extremely lengthy development time may not be the highest priority item because of the length of time to develop it. What happens if a story can't be estimated? You can split the story and perhaps gain more clarity. Sometimes splitting a story doesn't help though. If this situation occurs it may be necessary to do some research on the story first. Timebox the research! If you do not, it will take all available time thereby depriving the product of something else which could have been done instead.
- **Small**: Obviously stories are small chunks of work, but how small should they be? The answer depends on the team and the methodology being used. I teach agile and suggest two-week iterations which allow for user stories to average 3-4 days of work – TOTAL! This includes all work to get the story to a "done" state. Also remember not to goldplate user stories, if it meets the acceptance criteria development should stop.
- **Testable**: Every story needs to be testable in order to be "done." In fact, I like to think of testable meaning acceptance criteria can be written immediately. Thinking this way encourages more collaboration up front, builds quality in by moving QA up in the process, and allows for easy transformation to an acceptance test-driven development process.

FAQ

Why User stories and not Requirement Documents

Requirements are often:

- Not read as long documents are not very easy to use every day.
- Not usable as a unit of work (i.e. "The system should allow the user to chat with other users")
- Don't explain why certain features are needed
- Often jump into technical details, leaving no room for developers to do their job and select more effective solutions.
- Get very detailed and thereby replace more effective face to face team communication.
- Not prioritized completely, i.e. 10 requirements are "must have priority 1", thus creating large work packages and slow deliveries.
- In a rigid structure like a hierarchy in a document, which isn't easy to adapt to changes.

Further reading

Must read: https://blog.prototypr.io/stop-it-with-as-a-user-5feb9b38d920

https://www.adcisolutions.com/knowledge/why-write-user-stories

Improvement points

- Requirements writing could use it's own page (check what's already on acceptance criteria writing)
- Subtasks (only use when needed, ie 2 people on 1 story)
- Design User stories (user different ticket type for design user stories, that you convert to normal user story before development starts)
- Jira Boards and backlog filter for the dual track agile (doing design before dev).