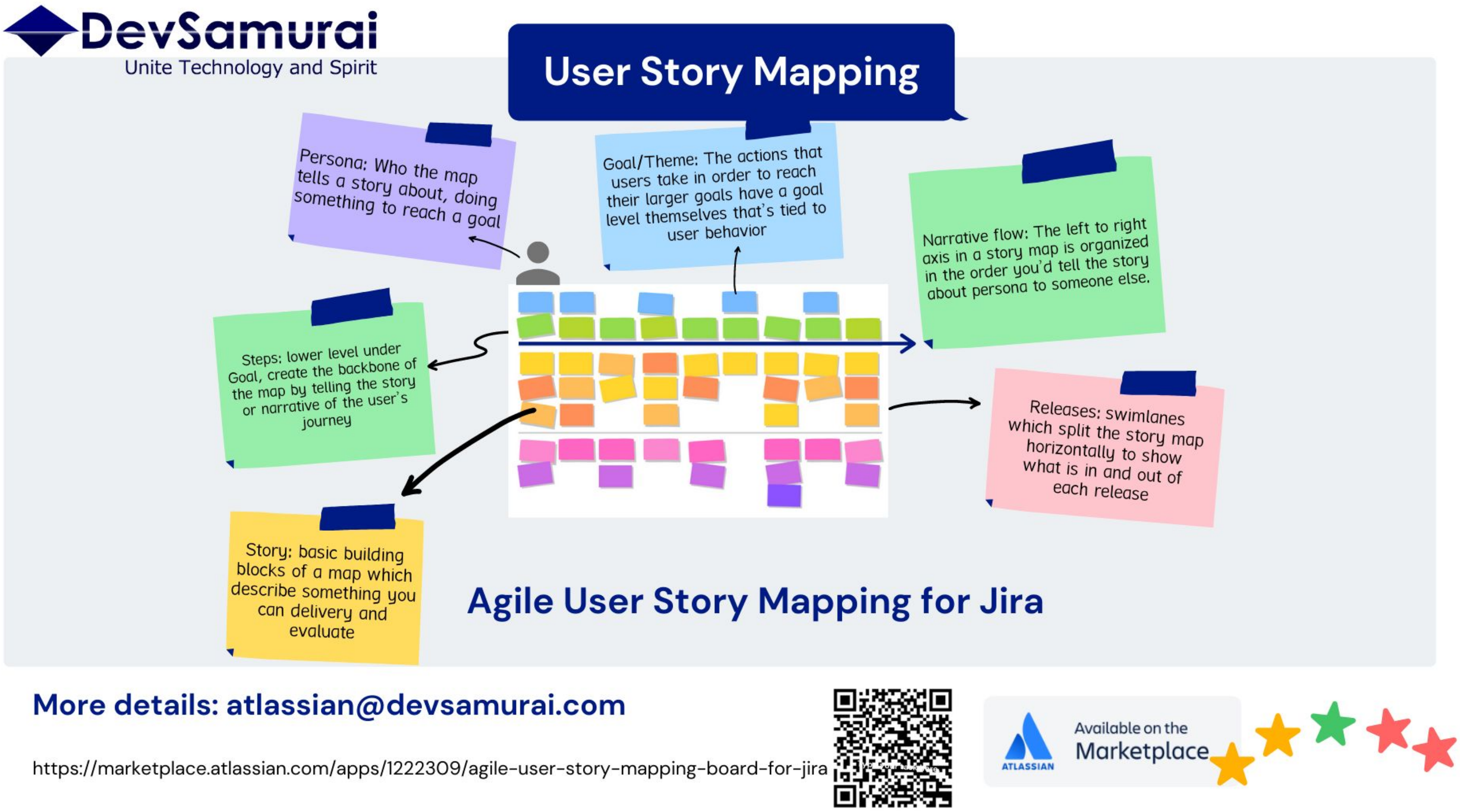
Agile User Story Mapping

DevSamurai
Unite Technology and Spirit

Contents



DevSamurai

Unite Technology and Spirit

User Story Mapping

Persona: Who the map tells a story about, doing something to reach a goal

Goal/Theme: The actions that users take in order to reach their larger goals have a goal level themselves that's tied to user behavior

Narrative flow: The left to right axis in a story map is organized in the order you'd tell the story about persona to someone else.

Steps: lower level under Goal, create the backbone of the map by telling the story or narrative of the user's journey

Releases: swimlanes which split the story map horizontally to show what is in and out of each release

Story: basic building blocks of a map which describe something you can delivery and evaluate

Agile User Story Mapping for Jira

More details: atlassian@devsamurai.com

https://marketplace.atlassian.com/apps/1222309/agile-user-story-mapping-board-for-jira

An Introduction to Agile Product Management

TASK 1
LOREM IPSUM DOLOR SIT AMET, PER SCRIPTA LAOREET
TASK 2
OPORTERE AT, NAM CU POSTEA

Agile software development approaches recommend that rather than building the full stated requirement, that just enough is built to allow the customer or stakeholder to interact with the system so that they can review the work completed and reconsider the next steps. This is called iteration.

What is Product Management?

Originally coined in 1995 by Agile Manifesto founder Ken Schwaber, product management is a concept which formally acknowledges that key software systems should be treated as ongoing and evolving endeavours with a focus on value (products) rather than one-off deliveries with a focus on delivering against an unchanging plan (projects).

Product managers facilitate this process by maximising the value delivered by their product and they do this by prioritising and pursuing the items that are likely to yield the most value when balanced against the effort to implement.

Product managers and product owners work closely with a variety of stakeholders, both internally (employees) and externally (end-users and third parties) in order to understand not only what they say they want (requirements), but to form theories and hypotheses about what they might need and not yet know (experimental test and learn).

An Introduction to Agile Product Management

Capturing Requirements

Part of being a product manager is speaking to stakeholders to understand what behaviours they want the system to exhibit and how they want it to function. This is where we get the term “**functional requirements**”.

Functional Requirements

Functional requirements are so-called because they are functions required by someone, usually stakeholders, and the requirement is related to the system being built. An example would be “I want to be able to log in to the system”.

Product managers are often subject matter experts regarding the product they are managing, so they may often write their own requirements. In such circumstances, they need to be careful that their requirements are validated with research otherwise there is the danger that the product becomes built to their own subjective requirements rather than externally validated ones.

Agile software development approaches recommend that rather than building the full stated requirement, that just enough is built to allow the customer or stakeholder to interact with the system so that they can review the work completed and reconsider the next steps. This is called iteration.

Non-Functional Requirements (NFRs)

When requirements relate to technical requests, these are called “non-functional requirements”, which are specifications for the system that do not directly affect the user-facing behaviour. An example of this would be “User passwords should be encrypted to the AES 256 standard”.

Non-functional requirements are very important as they relate to the security, performance, scalability and integration of the system being built, but it is easy to forget non-functional requirements until it is too late and that’s why it’s common to see large I.T. projects fail; it’s not that the user-facing design was wrong, it’s that insufficient rigour was invested into specifying the bits that you don’t see.

It is also common to see the reverse of the above issue where systems are over-engineered in regard to the NFRs. This usually happens because either people have taken an overly risk averse approach to the NFRs (e.g. stating that a system must be available 99.999% of the time when it’s only required during working hours) or they’ve used an example set of NFRs from a different project.

With the above points in mind it’s important to ensure that NFRs are “right-sized” and appropriate for the product being built to prevent failures which might impact business continuity, but also prevent overspend and overengineering, which can carry a burdensome resource load.

Product Requirements Document

When requirements are being gathered they are often captured in a product requirements document (PRD), which is a list of the stated requirements along with explanations of what they are and why they are required.

This list of requirements can then be used to write a series of actionable backlog items for the development team to start implementing. Such backlog items are often written as “user stories”.

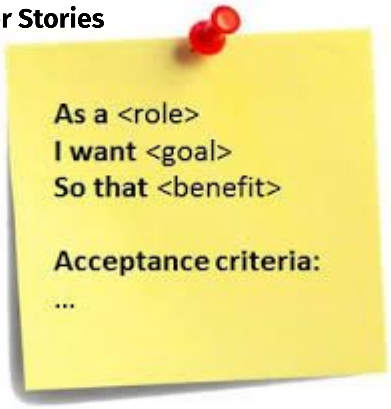
An Introduction to Agile Product Management

User Story Mapping

From the PRD we can then begin to start writing our list of user stories, and a useful approach to take is a user story mapping approach as follows:

Vision → Goals → Activities → Tasks → User Stories

It often helps to use the separate steps of the customer journey as an aide memoire to writing the user story map as it ensures no element is left out.


As a <role>
I want <goal>
So that <benefit>

Acceptance criteria:
...

User Stories

User stories, introduced by the e**X**treme **P**rogramming (XP) movement, are a way of representing requirements in a series of simple statements. The structure is as per their name in that firstly, the requirement must be linked to an end-user and secondly, they must be written as a story, rather than a solution.

An example of a good user story	An example of a bad user story
As a *shopper* ***I want to*** *search for cheap items* ***So that*** *I can save money on my shopping* This is a good example because it lets a person reading the story understand who the user is, what they want to achieve and why they want to achieve it. Further discussions can now occur as to how this story could be delivered. For instance, you could allow sorting of search results from lowest to highest, or you could have a quick link which displays a page containing items priced under $5. The point of this is, there are multiple solutions to a problem and if you jump to a solution too early then there is the risk that you choose the wrong solution.	***As a*** *Product Owner* ***I want to*** *optimise the DB by changing select queries to limit queries* ***So that*** *it performs faster* This is a bad example because firstly, **a product owner is not a user**, so we don't understand which user persona we are directly benefiting by completing this task (unless you happen to be building a system for Product Owners!). Secondly, the approach to resolution has been **pre-solutioned** without any room for negotiation and thirdly, the outcome **doesn't really state how the task adds any business value**, as a faster database in itself does not create value.

User stories: A reminder to have a conversation

A good user story is a reminder to have a conversation with your users, stakeholders and development team about how to solve the stated problem (not a reminder that a conversation has already occurred!). In fact, the process of implementing user stories is stated to be "The three Cs" as follows:

① **Card** – The writing of the user story

② **Conversation** – The discussion about how to achieve the desired outcome

③ **Confirmation** – The process of confirming that the outcome has been achieved

It is common, however, to see user stories written with full solutions attached before the development team has seen the item and this tends to be reflective of organisations who have moved towards agile from a more top-down waterfall approach. This isn't necessarily a catastrophic problem, but certainly better resolutions are achieved when the people implementing them are correctly engaged.

Workflow tools and Agile

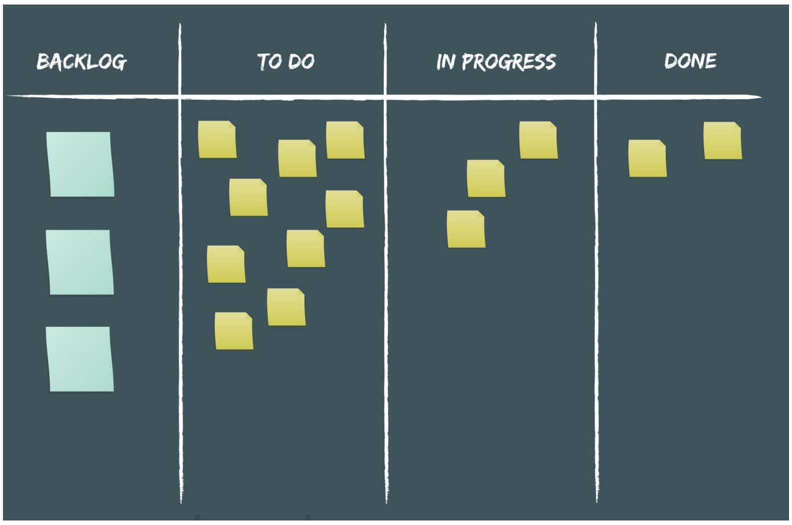
Originally, agile software development teams managed their workload using whiteboards known as "information radiators" which were separated into columns with sticky post - it notes to represent user stories and indeed some teams still operate like this today.

The purpose of the information radiator is to provide transparency, both within the team and outside the team so that the status of progress on an item can easily be known. Whether it is in progress, done or not yet started; it is important that people with a vested interest can make themselves aware of this progress and preferably without having to disturb the developer by asking "is this done yet?".

With the advent of distributed teams and remote working, there was a move towards virtual team boards, stored inside computer-based workflow tools. **Some popular workflow tools include "Jira" and "Trello" both owned by Atlassian and also "Azure Devops" by Microsoft.**

Such workflow tools allow you to create detailed backlog items, containing not only the original problem statement or user story, but all of the associated conversations, images, process flows, links to relevant code snippets and much more. These items then move across a virtual whiteboard from "To Do" all the way to "Done".

A happy side effect of using virtual workflow tools is that you have a detailed audit record already completed showing how long an item took, who requested it, who worked on it, what the solution was and even a link to the specific code that the developer committed to their repository. There have been some criticisms of workflow tools, namely that they change the focus of the team to spend too much time on administering the tool itself and that the way in which some teams work then begins to change to reflect the language and default workflow of the tool itself.



With that in mind, it's important to ensure that the team is working in the best way for itself and the organisation, by ensuring that the scrum master or agile coach working with the team, guides them to make appropriate choices about ways of working.

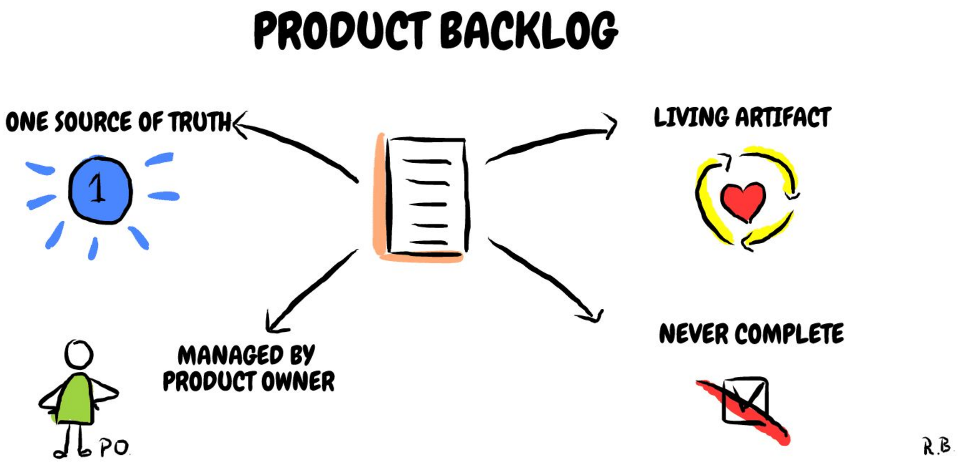
An Introduction to Agile Product Management

Agile Product Backlog

Once the PRD has been distilled into a set of user stories, these stories need to be stored somewhere in priority order. The name for the place that this to-do list is stored is the “Product Backlog”. It’s the product manager or product owner’s responsibility to ensure that the product backlog is in priority order so that the team knows which items will create the most value.

Once the backlog has been created, further discussions can begin with the team in a process known as “refinement”. This allows the necessary detail to be added to user stories to allow the development work on them to proceed.

Although the product manager specifies the priority order of the backlog, the team is free to choose items from it in the way that best makes sense, as they are better informed with regards to the dependencies involved in the development process.

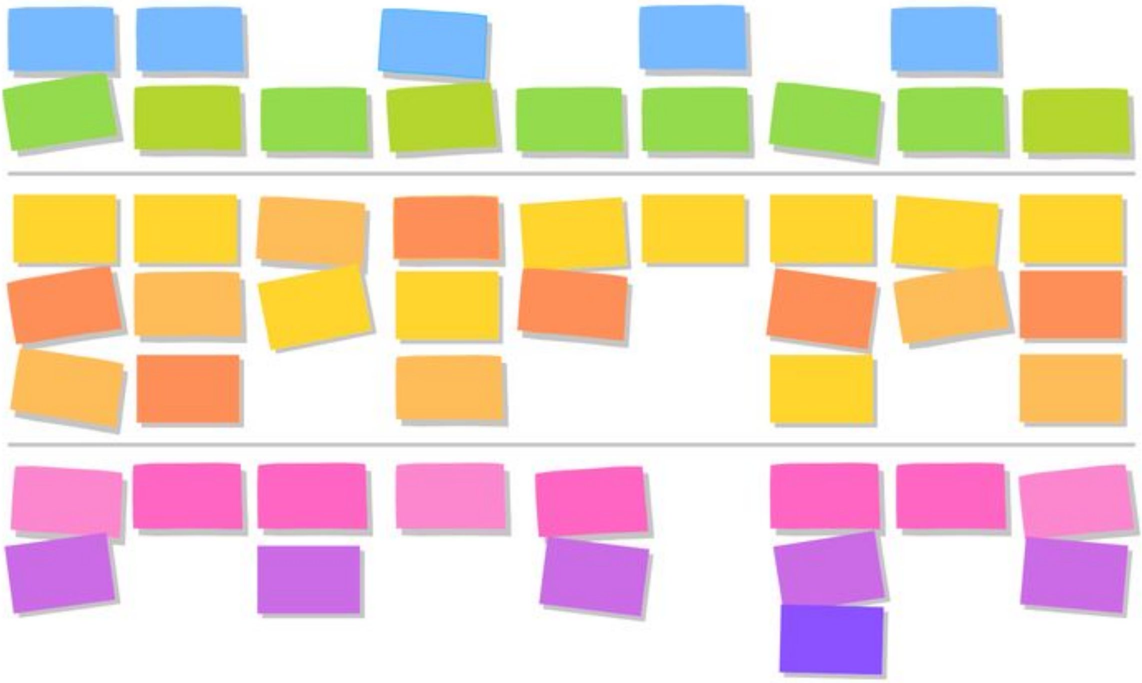


Product management is a complex area, ensuring that the views of all relevant stakeholders are considered, prioritised, assessed for value and then translated into actionable development items. Product managers need to be able to speak to people from a diverse range of backgrounds such as board level directors, users and developers.

As with everything in agile, this is an iterative process where improvements are made along the way, both to the engagement process with stakeholders as well as the development of the product itself.

By using our Jira plug in you can help smooth the product management process by providing an easy to read view of your user stories and user story map, including the progress and priority of each one.

What is User Story Mapping?



A common challenge for many organizations is how to take the high level stated requirements from senior staff, stakeholders and users and translate them into actionable and logically ordered items.

Even if you write a long and detailed list it can be very difficult to read, consume and check for omissions. Many people are visual learners and so respond well to a visual picture.

Why use User Story Mapping?

In traditional waterfall projects, lengthy requirements documents would be prepared, which development teams would work against until all stated sections were deemed to be met. At this point the project would move into a testing phase and then a user acceptance testing phase. Although these documents can be useful in terms of providing a large amount of information, there are some drawbacks to their inflexible and unwieldy nature.

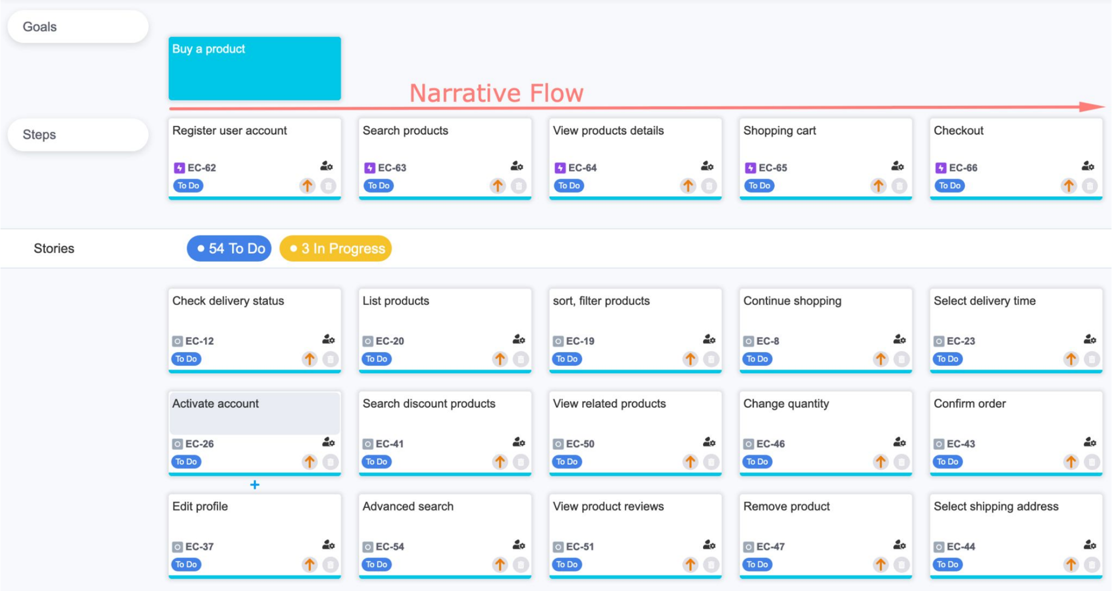
User stories have been widely accepted as **the de-facto standard for conveying requirements to development teams**, but just a long list of user stories can be a bit confusing to those consuming it, both from a stakeholder perspective and from a developer perspective.

User story maps provide **an easy to view, structured, transparent and contextual view of user stories ensuring that any of the required steps for the journey have not been missed.**

What is User Story Mapping?

How to do User Story Mapping?

The user story mapping process consists of the following steps



01. Identify your high level activities

This can be done by reviewing the user journey for the relevant part of the system that you want to create the map for. **Activities** are also known as “**Goals**” or “**Themes**”. **How?**: Such journeys should be relatively clear and apparent from the high level requirements, but should be reconfirmed with the relevant stakeholders. Once you have this list of activities you arrange them in chronological order from left to right as if you were telling the story of your user’s journey

02. Document the steps

Document the **steps** required to complete all the necessary functionality in order to achieve the **activities**. These **steps** are often individually labelled as **epics** as they tend to encompass closely linked groups of user stories and tasks and are often referred to as the “**backbone**” of the story map, as they provide much of the structure.

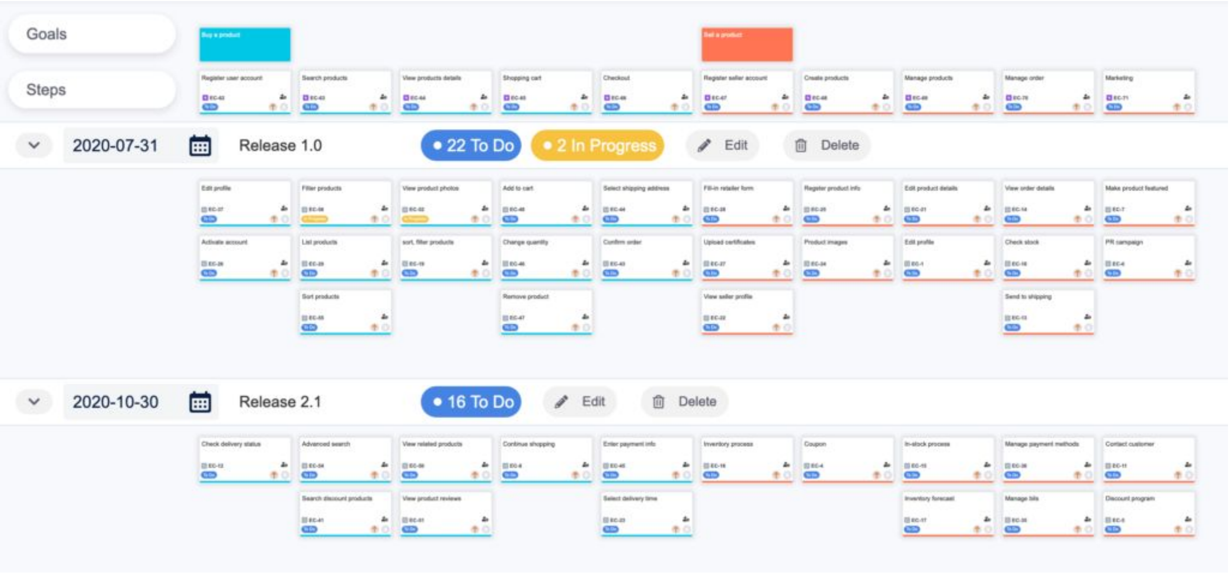
03. Identify the details

These will be the actual user stories and tasks required at a fine granular level. These are best fleshed out in conjunction with not only the stakeholders who are asking for the functionality, but also with the team who will be building it to ensure the best solutions are designed.

Once you have the map laid out you can start identifying natural slices of the product that would be suitable as user releases. The best way of developing software involves continuous integration so that software is integrated and released in a little and often fashion, but just because the code has been merged to the code base it doesn’t mean you have to make every small change available to the user immediately.

What is User Story Mapping?

Example of User Story Mapping



Screenshot from DevSamurai Agile User Story Map for Jira

The below image shows an example user story map covering elements of a user log in page and changing of account details for an e-commerce site. Note that the blue boxes map to **activities**, the orange boxes to **steps** and the grey boxes more to the **details** of how the specific items will be achieved.

When is User Story Mapping Not Suitable?

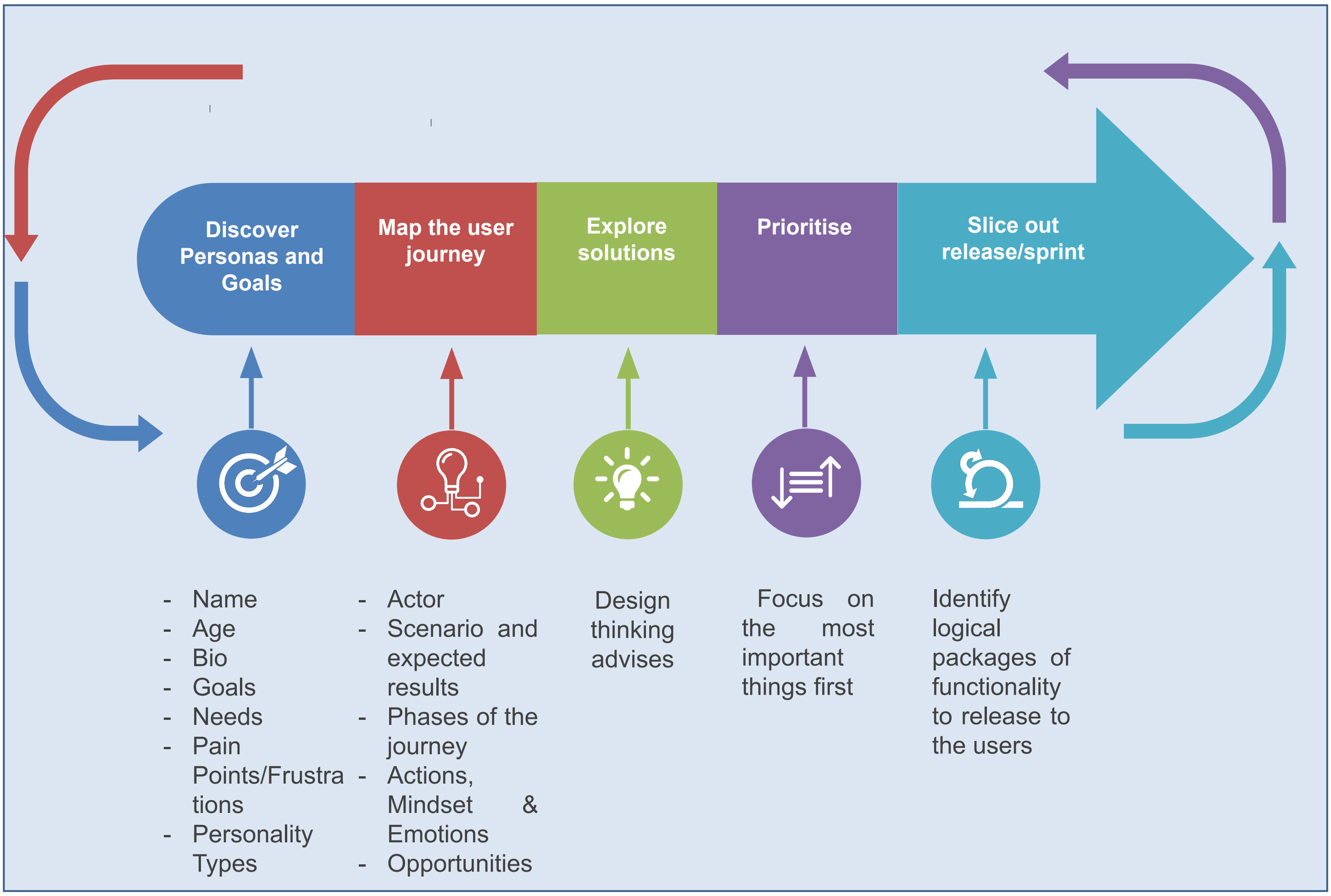
User story mapping won't work in every situation. If there is no clear user journey, such as with API building work then it would be quite difficult to use. If your team was doing more operational response, such as fixing bugs or responding to user issues, then it would not be possible to create a user story map for such work.

In a highly experimental agile environment where the next iteration's work was highly dependent on the outcome of the experiments from the last release, such as a startup in the discovery phase, extensive user story mapping would also not be possible, although you could at least use it to plan the next increment of delivery.

User story mapping is a highly interactive and visual method of representing a list of stated requirements from stakeholders, senior employees and users of the system in a chronological and structured way.

It provides a highly transparent way of examining the functionality which has been identified to the build, the order in which to build it and the desired release slices that will be made available to the users.

User story mapping steps



Discover Personas and Goals

- Name
- Age
- Bio
- Goals
- Needs
- Pain Points/Frustrations
- Personality Types

Map the user journey

- Actor
- Scenario and expected results
- Phases of the journey
- Actions, Mindset & Emotions
- Opportunities

Explore solutions

Design thinking advises

Prioritise

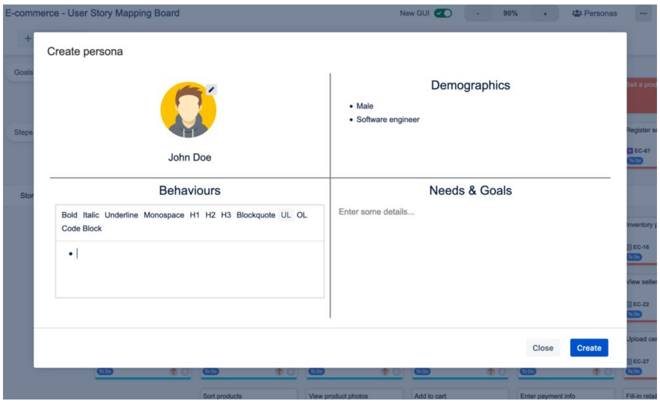
Focus on the most important things first

Slice out release/sprint

Identify logical packages of functionality to release to the users

Step 1: Discover Personas and Goals

A persona is a profile of an individual who may interact with the system in a unique or individual way. To ensure that all the relevant requirements have been considered and understood, an early activity will need to be conducted to identify and discover user personas. A user persona tends to be presented as a one-page card with information such as:



Create persona

John Doe

Demographics

- Male
- Software engineer

E-commerce - User Story Mapping Board
New GUI
90%
Personas
Behaviours
Bold Italic Underline Monospace H1 H2 H3 Blockquote UL OL Code Block
Needs & Goals
Enter some details...
Close
Create

User persona creation form – Agile User Story Map for Jira

- **Name** (An example name to reflect that particular persona)
- **Age** (An indicative age reflective of people who exemplify that persona)
- **Bio** (A short description about this persona)
- **Goals** (The things that they mainly want to achieve in life)
- **Needs** (The things they need to achieve from the system being built)
- **Pain Points / Frustrations** (A list of things that annoy or slow this user down)
- **Personality Types** (Are they outgoing, shy, reserved, detail-oriented?)

User story mapping steps

A persona is a profile of an individual who may interact with the system in a unique or individual way. To ensure that all the relevant requirements have been considered and understood, an early activity will need to be conducted to identify and discover user personas. A user persona tends to be presented as a one-page card with information such as:

Step 2: Discover Personas and Goals

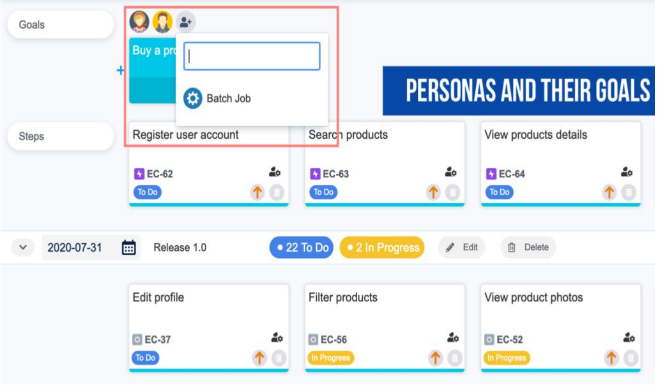
Once personas have been identified, they can each be reviewed in turn based on all of the things each person might want to achieve from the system. Once this full list of interactions with the product has been listed, they can be written down in a visual timeline known as a user journey map.

The user journey map is a visual representation of a customer's interaction with your product from start to finish, showing how they achieve the things that they need to and what steps they need to go through in order to reach this destination.

The best products optimise their user journeys to ensure that objectives can be achieved by their customers in as few steps as possible. The reason for optimising the journey is that there are many competing products on the market and customers will naturally gravitate towards those that make their lives as easy as possible.

Journey maps require the following components:

1. **Actor**: To which persona does this part of the journey map refer
2. **Scenario and expected results**: What objective is the actor trying to achieve and what outcome do we expect
3. **Phases of the journey**: which may include phases that are completed outside of the system. As an example – a user buying a wireless speaker from an ecommerce shop would first see the speaker advertised, then buy it, then take delivery, then try it and finally may even return it.



Map persona with the user journey – Agile User Story Map for Jira

4. **Actions, Mindset & Emotions**

- **Actions**: The specific behaviours and steps taken by users
- **Mindset**: users' thoughts, questions, motivations at various stages throughout the journey
- **Emotions** – a line showing highs and lows of the journey, so that these can be leveraged or improved as appropriate

5. **Opportunities**

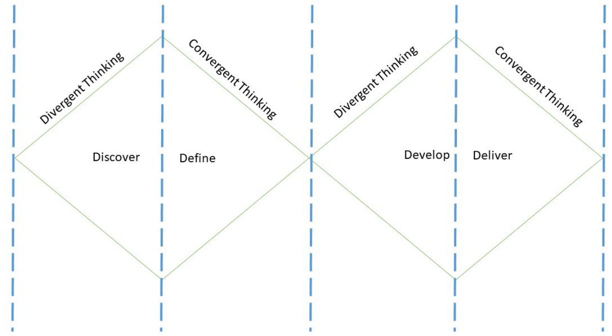
Armed with the above information, what do we need to do/change/improve and who owns which action? How will we measure whether a change has been made, what metrics will we use?

Once we have our identified journeys mapped out, built on our user personas, we can then start to identify the solutions we are going to put in place in order to satisfy these requirements.

User story mapping steps

Step 3: Explore solutions

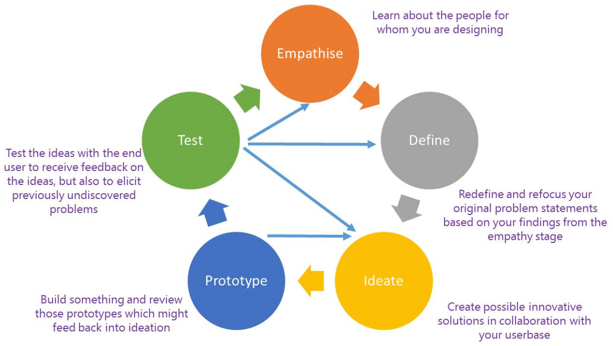
When exploring solutions a good approach to use is “Design Thinking”. Design thinking advises that the best designs come from casting the net wide initially, then focusing in on defined solutions, before developing and testing a wide range of solutions and finally, narrowing down to the winning



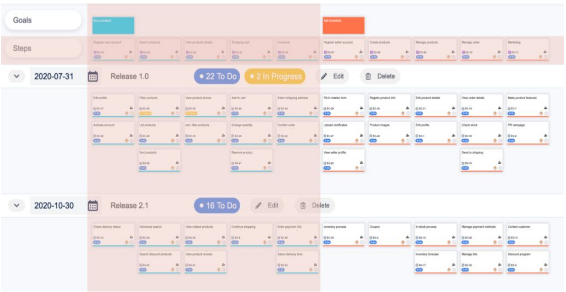
The above model was built on further by Plattner et al, and as can be seen below it is not a linear process with a defined beginning and end, but rather an iterative cycle of constant development and improvement. If we were starting with a brand new product, we may start at the ‘empathise’ phase, however, for pre-existing products we could start elsewhere such as ‘test’, ‘define’ or ideate.

So called “**winning experiments**” then become the chosen and preferred solutions to the challenges and opportunities posed by the research in the user persona phase (also part of the “empathise” phase of the above diagram).

In this way, we can ensure that we have well-researched and tested solutions, rather than just solutions that people from the project team have proposed. These can then be translated into user stories or product backlog items for the team to turn into working software.


Empathise
Learn about the people for whom you are designing
Define
Redefine and refocus your original problem statements based on your findings from the empathy stage
Ideate
Create possible innovative solutions in collaboration with your userbase
Prototype
Build something and review those prototypes which might feed back into ideation
Test
Test the ideas with the end user to receive feedback on the ideas, but also to elicit previously undiscovered problems

Step 4: Prioritise



To prioritise is to **focus on the most important things first**. In software development, there are a number of different ways of prioritising the work to be completed. Product owners or those focused more on the functional side of the project are likely to consider the things which generate value to be of the highest priority. However, developers and technical members of staff are likely to have a different view of priority, as they will focus more on the things that are likely to make the system robust, performant and easy to develop on.

User story mapping steps

One of the more popular development frameworks, “Scrum” advises that the **product owner should order** (prioritise) the backlog of work in a way that they find the most appropriate (in practice this tends to be by value), but that the development team may implement the work in the way that they choose. The development team is guided by the order of work, but not bound by it.

It is important that the development team is able to choose the order in which they implement the work because non-technical staff may not understand the things that need to be completed behind the scenes in order to make a fully functional product. Additionally, there may be a significant reduction in effort by delivering a high priority item with a low priority item, the so-called “whilst you’re under the hood” factor.

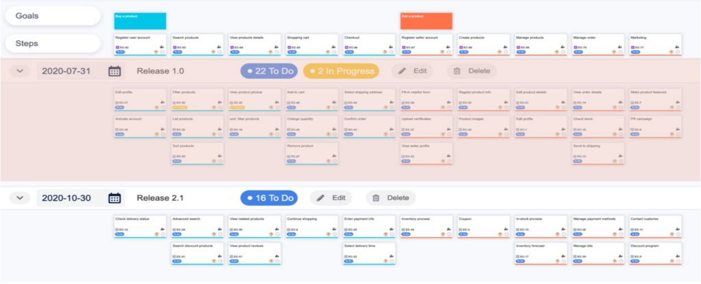
In true agile development, the most value is said to come from the work that is going to yield feedback the earliest. This means that there is value in even delivering a non-functioning button on a page if it allows for feedback to be received.

In practice, it depends heavily on what is being built, the culture of the organisation and user group and the appetite for experimentation as to the correct metric to use for prioritisation.

Step 5: Slice out development release/sprint

Once the personas have been identified, the journey has been mapped and the component deliverables have been ordered and prioritised in a suitable way with feedback from across the team, it is then possible to begin identifying logical packages of functionality to release to the users, otherwise known as “**release slices**”.

The personas, journey map and solutions can be translated into a set of deliverable items using **user story mapping** to create the lower-level tasks that the development team will turn into working software in order to build the product.



Slide the release in Agile User Story Map for Jira

Release slices can also be utilised as the **product goal** in Scrum teams, with the individual sprint goals all working towards that release. This allows teams to begin to forecast forthcoming sprints, using a high-level sprint goal per sprint, until everything is completed to allow the release slice to be made available to end-users. It should of course be noted that such forecasts are not a binding plan, and should not be represented to stakeholders as such.

Whilst development teams working to best practice aim for **continuous integration, delivery and deployment** to production as a way of ensuring good code quality, it may not always be suitable to make the latest functionality available to users immediately. This is because when building a new phase of a user journey it often does not make sense to release the functionality until the full phase of the journey has been completed.

Development teams can balance the need for continuous deployment as a good coding practice, with the requirement to only release chunks of functionality, by using **feature flags**. Feature flags allow code to be released to the production environment and tested but not actually visible to end users until the feature flag is enabled. When the flag is enabled this requires no release or coding, merely a flag to be toggled and then the functionality becomes available, which is how many “releases” are completed.

About DevSamurai

“ DevSamurai, Inc is an IT services firm based in Japan and APAC. We provide DevOps, system development services and solutions such as Robotic Process Automation (RPA), AI Chatbots to automate business and IT processes

We help customers to transform IT to next level with latest cloud computing platform, devops tools and best practices. Our team provide industry leading consulting expertise, service delivery, cutting edge products and solutions to all steps of Software Development Life Cycle (SDLC). Our Robotic Process Automation (RPA) and Chatbots solutions also empower organizations to automate business processes. We free customers from repetitive tasks to only focus on producing valuable outcomes.

DevSamurai
Unite Technology and Spirit

User Story Map

Member of

CLOUD NATIVE COMPUTING FOUNDATION

THE LINUX FOUNDATION

Find us on Atlassian Marketplace

Contact Us

📍 58-6 Oyaguchikitacho, Itabashi-ku,
Tokyo 173-0031 Japan

📍 4F Shiroyama Trust Tower,
Toranomon 4-3-1, Minato City, Tokyo,
105-6004 Japan

✉ info@devsamurai.com

☎ 050 362 78910

🌐 https://www.devsamurai.com/