3. Agile Approach

3.1. Methodical Approach

Originally proposed by Beck[1] in 1999 and refined by various authors, *user stories* are natural-language approach to expressing requirements from the point of view of a specific user of the system. User stories are kept in a *backlog* in order of priority.

Patton[2] has criticized that backlogs, like all requirements documents, tend to be flat lists and has instead proposed the *user story map* as a way of structuring requirements. A user story map is a hierarchical collection of requirements that are derived from *user activities.* User activities describe objectives that a user may have and can be further broken down into *user stories.*

Patton further argues that each system has a number of requirements which are so essential that not the system cannot function if they are missing – the *backbone*. Because missing just one of the requirements from the backbone renders the system unusable, these requirements cannot be further prioritized.

The backbone can be decomposed into smaller user stories however, some of which may be more critical than others. A small number of user stories is usually enough to satisfy the minimum of the backbone requirements. Because these stories in principle allow the system to function, Patton calls them the *walking skeleton* (see Figure 1). This walking skeleton corresponds with the concept of a *minimum viable product*, which, while not ready for productive use, already covers the core functionality of the intended system.

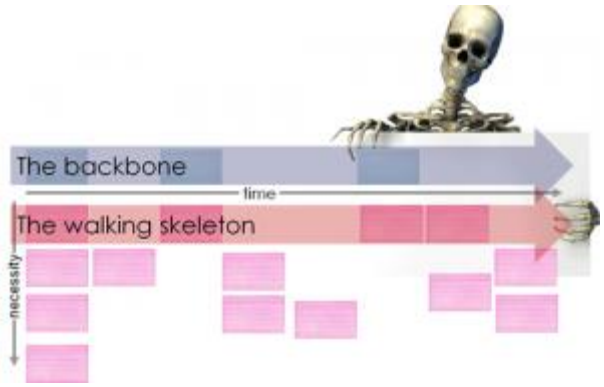

Figure 1: User Story Maps feature essential requirements in the backbone. These requirements are then decomposed into user stories. The user stories most essential for delivering the backbone are also called the "walking skeleton" (source: Jeff Patton[3]).

Subsequent user stories will then add to the functionality in increasing levels of granularity. This creates a hierarchy of user stories, where the stories most critical for delivering the backbone must be implemented first (see Figure 2). Blue boxes represent the backbone, green boxes represent use cases that make up the walking skeleton and yellow boxes are additional use cases.

---

[1] Beck, Kent (1999): Extreme Programming Explained: Embrace Change. Addison-Wesley.
[2] Patton, Jeff (2005): It's all in how you slice it.
[3] Patton, Jeff (2008): The New User Story Backlog is a Map. Retrieved from https://www.jpattonassociates.com/the-new-backlog/. Last access on 2021-06-02.

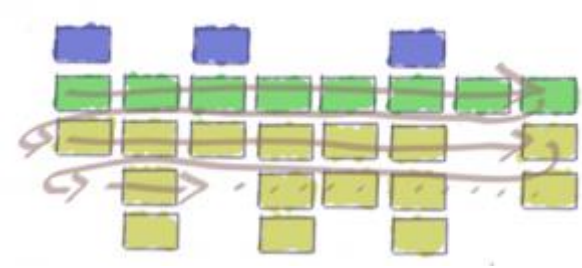

Figure 2: A user story map visualizes which user stories are absolutely critical to delivering the product, even if they are spread across multiple features. (source: Jeff Patton[4]).

3.2. User Story Map

Figure 3 depicts how the requirements outlined in chapter 4 might be structured in a user story map. The blue boxes describe user activities, sometimes called epics or user journeys. The yellow boxes are the backbone of the system and consist of comparatively large user stories that are necessary for using the system. The white boxes are the smaller-scale user stories that collectively form the backbone. The walking skeleton is called *MVP (Minimum Viable Product)* in this map – these are all of the user stories that are strictly required for essential use of the system. Cagan claims that because the walking skeleton is intended as more of a tool for validation and testing rather than productive use, the term *minimum viable prototype* may be more adequate.[5]

[4] Patton, Jeff (2008): The New User Story Backlog is a Map. Retrieved from https://www.jpattonassociates.com/the-new-backlog/. Last access on 2021-06-02.
[5] Cagan, Marty (2017): Inspired. John Wiley & Sons, 2. Edition.

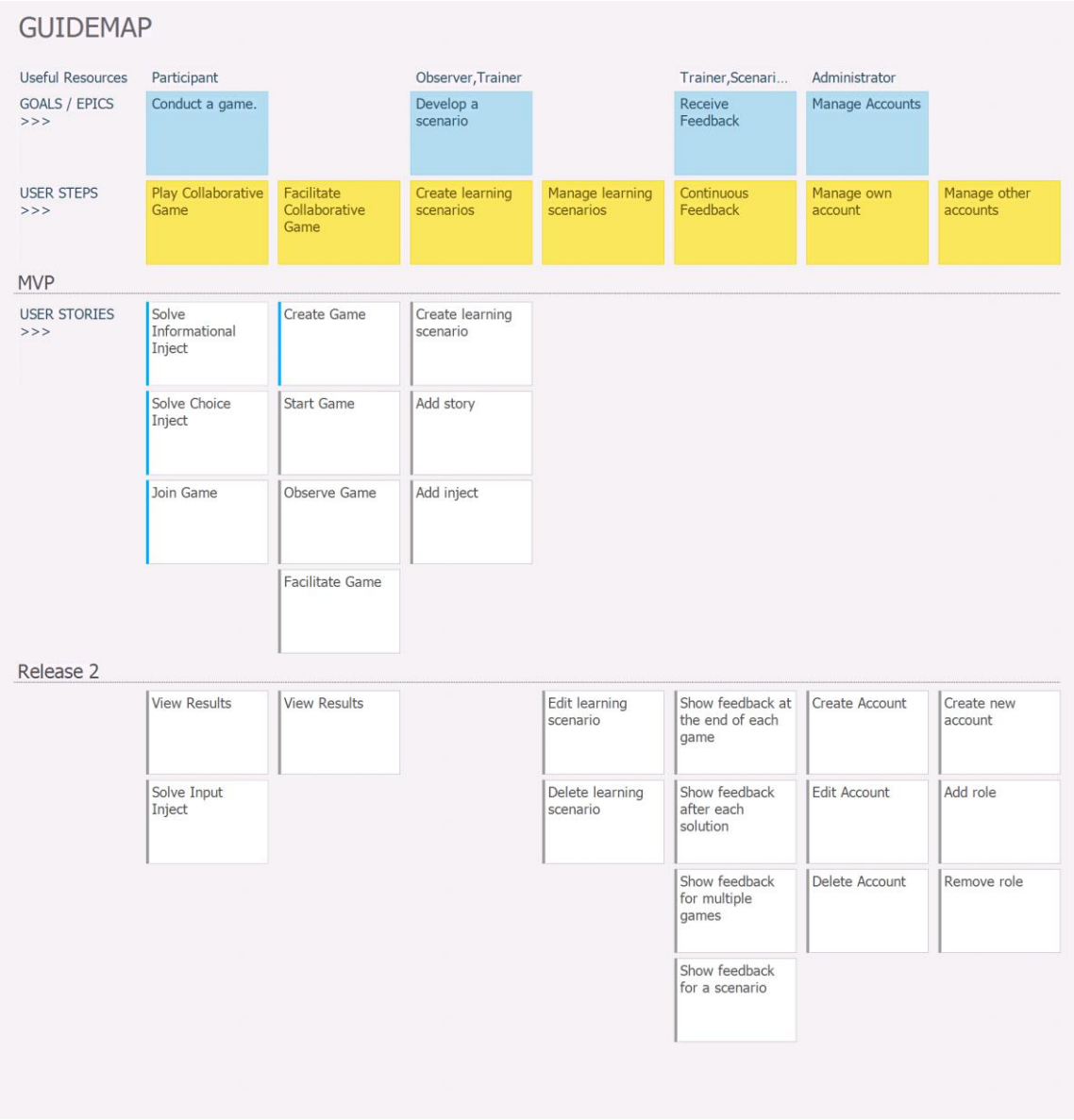

Figure 3: The user story map of the planned system (source: own work).

As is apparent from Figure 3, the user journeys “Play Collaborative Game”, “Facilitate Collaborative Game”, “Create Learning Scenarios”, “Manage Learning Scenarios” and “Continuous Feedback” must be completed to at least some extent for the release of the MVP.

The status of each of the user stories is provided by the colored line at the left. The blue line for “Solve Inject”, “Join Game” and “Create Game” indicate that these stories are currently in progress, whereas grey lines represent user stories that are still open.

3.3. Detailed Descriptions of the User Stories

A user story map provides an excellent overview over the existing requirements and their priority with respect to the entire project. Nonetheless, more information is required to enable actual implementation and testing of these user stories. Therefore, the user stories shown in Figure 3 will be described in more detail in this chapter. The user stories follow the Connextra Template[6].

[6] Agile Alliance (ed.) (2021): User Story Template. Retrieved from https://www.agilealliance.org/glossary/user-story-template/. Last access on 2021-06-04.

Figure 4 describes a user story from the perspective of a game participant. It has a descriptive title (“Solve Informational Inject”) and a short description of the requirement (“to solve an informational inject”). This particular story additionally comprises a definition of done (DoD), which is a checklist of hard requirements which the system must pass if this story is to be considered as “done”.

The user story also has a feature test written in the “GIVEN … WHEN … THEN …” template, which describes the feature test that will verify whether this user story functions properly.

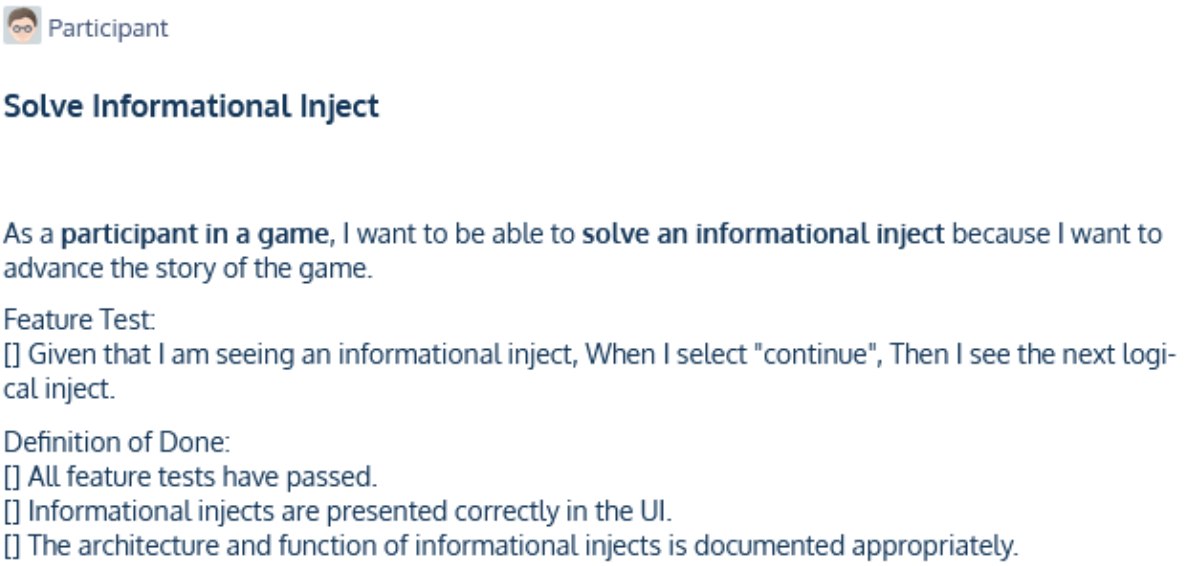

Figure 4: The user story "Solve Informational Inject".

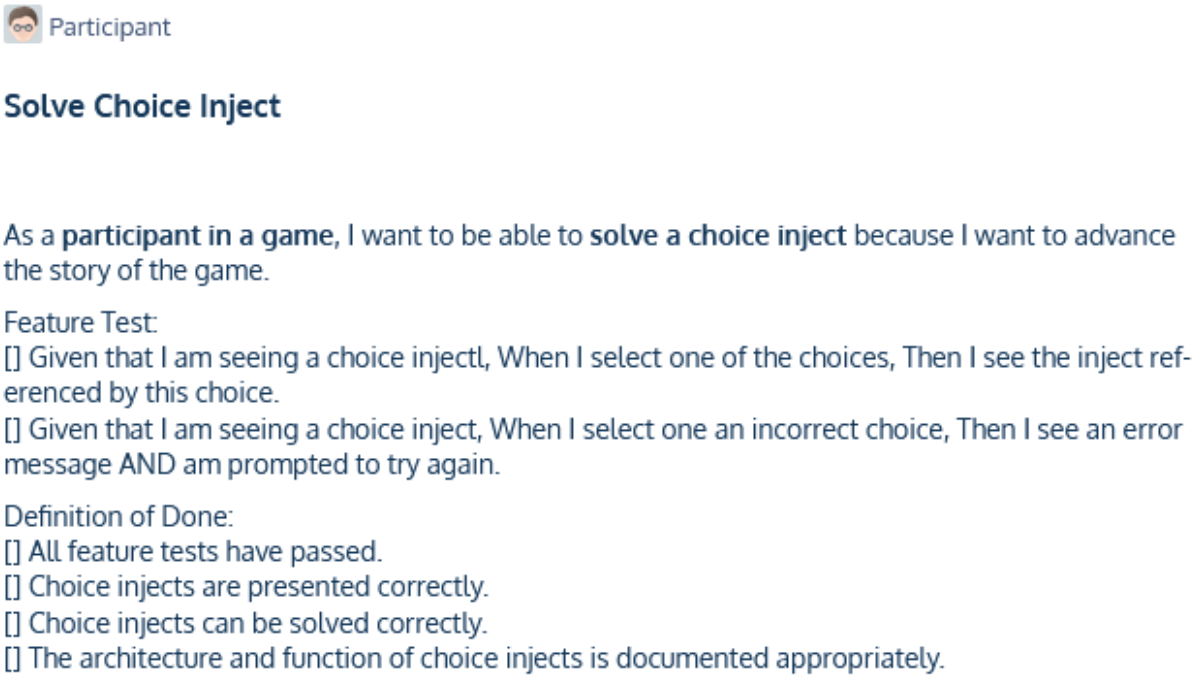

Figure 5: The user story "Solve Choice Inject"

In addition to the obligatory title, story description and DoD, the story “Join Game” (see Figure 6) also has the tag “Depends on other”, because it has a logical dependency on the story “Create Game”.

Depends on other

Join Game

As a **participant** I want to be able to **join an open game**, because I want to be able to play it.

Feature Test:
[] GIVEN that there is at least one open game AND that I am allowed to access this game WHEN I select "join" the game THEN I am registered as a participant for this game.

Definition of Done:
[] All feature tests have passed.
[] It is possible to perform this user story through the UI.
[] The code and functionality is adequately documented.

Figure 6: The user story "Join Game".

Create Game

As a **Trainer** I would like to be able to **create a game** so that participants can play it.

Definition of Done:
[ ] At least one test scenario exists.
[ ] A game can be created with the test scenario.
[ ] Participants can join the test scenario.

Figure 7: The user story "Create game".

Depends on other

Start Game

As a **trainer** I want to be able to **start a game** so that participants can participate in my training session.

DoD:
[] The newly started game can be joined by participants (see story "Join Game")

Figure 8: The user story "Start Game".

Observe Game

As an **observer**, I want to be able to **observe a game**, because I want to be able to evaluate the effectiveness of the training.

DoD:
[] It must be possible to see on which inject participants are currently working on.
[] It must be possible to see which variables exist in the game.
[] It must be possible to see what value each of the variables has.
[] It must be possible to see any other injects and stories.

Figure 9: The user story "Observe Game".

Depends on other

Facilitate Game

As a **trainer**, I want to be able to **facilitate a game** so that participants have an optimal learning experience.

DoD:
[] A trainer can adapt the challenge level of a game by (de-)activating certain injects
[] A trainer can change the global variables of a game
[] A trainer can abort a game.

Figure 10: The user story "Facilitate Game".

Create learning scenario

A a **scenario designer** I want to be able to create a new learning scenario.

Definition of Done:
[ ] I can create a learning scenario through a graphical interface.
[ ] The learning scenario is persisted.
[ ] A new learning scenario must have at least a name and one story.

Figure 11: The user story "Create learning scenario".

Depends on other

Add story

A a **scenario designer** I want to be able to **add a story** to a learning scenario, because they are a unit of coherent narrative.

Definition of Done:
[ ] I can create stories that are connected to an existing scenario.
[ ] I can do this through the frontend.
[ ] I have a class diagram that explains the relationship between story and scenario.

Figure 12: The user story "Add story".

Depends on other

Add inject

A a **scenario designer** I want to be able to **add an inject** to a story, because this allows me to develop the scenario.

Definition of Done:
[ ] I can create injects that are connected to an existing story.
[ ] The new inject must have at least a title and a description.
[ ] The new inject will be persisted automatically.
[ ] I can do this through the frontend.
[ ] I have a class diagram that explains the relationship between inject and story.

Figure 13: The user story "Add inject".