Critical Thinking Assignment 2: Agile Methodology in Software Development

Alexander Ricciardi

Colorado State University Global

CSC470: Software Engineering

Dr. Vanessa Cooper

December 22, 2024

Critical Thinking Assignment 2: Agile Methodology in Software Development

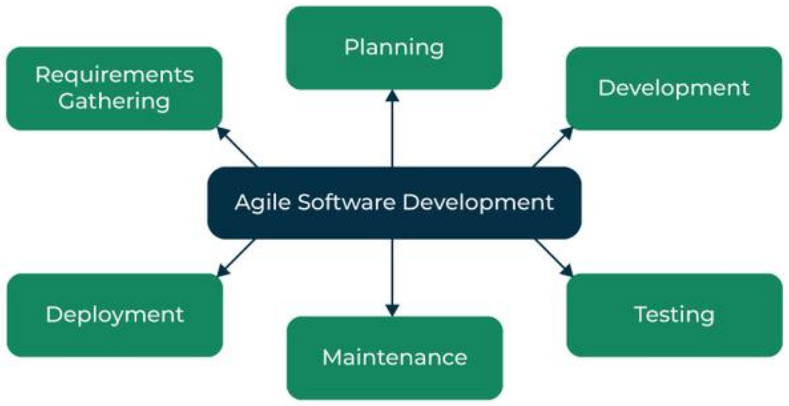
Agile methodology is a popular project management model used in Software Development (SD) that prioritizes flexibility, collaboration, and customer-centricity (Paliwal, 2024). Approximately 71% of U.S. companies have adobted the Agile methodology to resolve various business issues, to the point where "Agile is becoming a non-negotiable option for businesses across the globe" (Beckman, 2024, p.1). Furthermore, major tech. companies practice Agile in their software development process; companies such as Apple, Microsoft, Google, Amazon, Netflix, Spotify, PayPal, IBM, Cisco, PlayStation Network, and Adobe to cite a few. Therefore software engineers need to have a good grasp of Agile principles. This essay provides a basic description of the Agile methodology in Software Engineering, explores how Agile practices can be used in gathering and modeling nonfunctional requirements, lists two advantages and two challenges in modeling requirements with user stories in an Agile approach, provides Agile practices examples in system design, and examples of Agile roles, artifacts, and ceremonies.

Basic Description of Agile Methodology in Software Engineering

In Software Engineering, Agile is an SD model that focuses on delivering smaller incremental pieces of work, rather than waiting for a single, large-scale release (Paliwal, 2024). In other words, it is a methodology that, unlike traditional waterfall methods, which follow a rigid sequential process, Agiles is a flexible and incremental approach to SD; it is "a methodology based on constant improvement" (Petrescu, & Sterca, p. 1). The methodology process components also called phases are requirements gathering, planning, development, testing, deployment, and maintenance; see Figure 1.

Figure 1

Agile Software Development Components



Note: From "Agile Software Development – Software Engineering" by Naidu (2024)

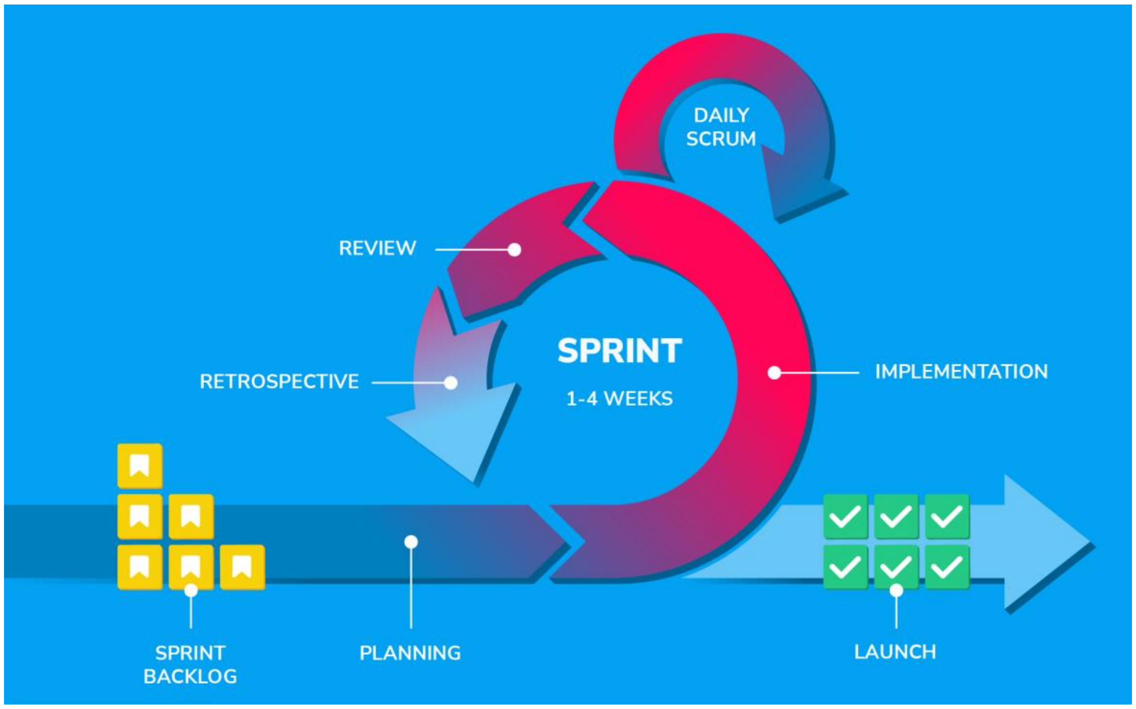
These components can be defined as follows:

1. Requirements Gathering: The process of gathering the customer's requirements for the software (Naidu, 2024).
2. Planning: The process of creating a plan for delivering the software, including the features that will be delivered in each iteration.
3. Development: The process of working to build the software, using frequent and rapid iterations called sprints.
4. Testing: The process of thoroughly testing the software to ensure that it meets the customer's requirements.
5. Deployment: The process of deploying the software.
6. Maintenance: The process of maintaining the software to ensure that it continues to meet the customer's needs and requirements.

Agile can also be described as having a time-bound, iterative approach to software management and development that incorporates a cyclical aspect to each of its components/phases.

Figure 2

Scrum Sprint



Note: This illustrates a sprint. A sprint is composed of implementation, review, retrospective, and a daily scrum team meeting. From "The Guide to Sprints in Scrum" by Szmit (2023).

Each of these phases can be carried out in a series of multiple short cycles. Within the Agile Scrum framework, these are defined as sprints, which incorporate various sprint sections and events, each with a specific goal (Szmit, 2023). See Figure 2 illustrating a sprint. A sprint is a short, time-boxed period during which a Scrum team works to complete a defined amount of work. They also enable rapid adjustments and frequent reassessment of project goals (Nguyen et al., 2024). Additionally, Scrum is just one type of Agile framework; other frameworks can be implemented depending on the specific requirements of an organization, such as Kanban, Lean Software Development (LSD), Extreme Programming (XP), Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), Crystal, Scaled Agile Framework

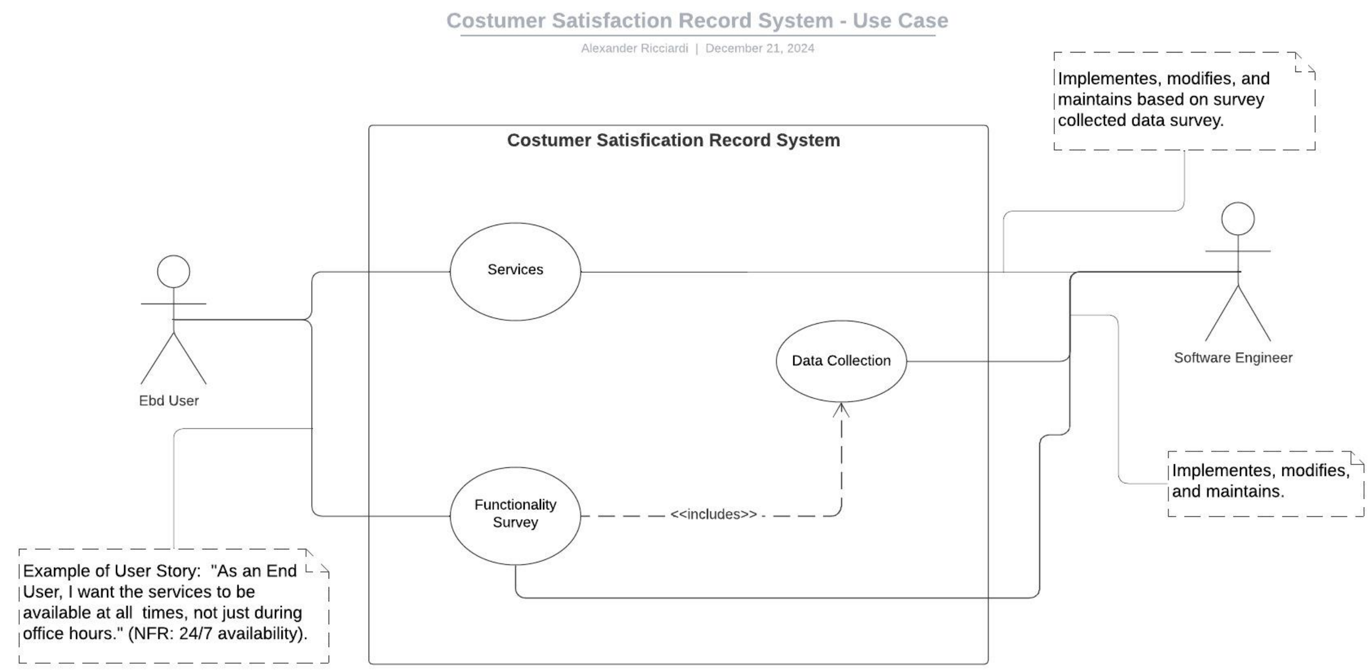
(SAFe), Adaptive Software Development (ASD), and Rapid Application Development (RAD) (Saraev, 2024). However, Scrum is the most popular framework in SD. Nonetheless, the iterative and cyclic approach to management and development is the soul of all Agile frameworks, allowing for frequent feedback and adjustments. “This iterative approach enhances alignment with customer needs and fosters open communication and shared responsibility within teams” (Nguyen et al., 2024, p. 3). With this understanding, let’s visit how Agile practices can be used in gathering and modeling non-functional requirements.

How Agile Practices Can Be Used in Gathering and Modeling Non-functional Requirements

Non-functional requirements (NFRs) define the quality attributes of a software system (Mazurkiewicz, 2023). They typically address different parts of a software system like availability, reliability, performance, stability, security, compliance, compatibility, accessibility, and usability. Their main purpose is to make a system more efficient and convenient to use. Agile methodologies by their iterative, colorative, and adaptable nature, if used correctly, provide a structure for gathering and modeling NFRs. Additionaly, this colorative and iterative nature of Agile allows for the continuous identification, gathering, and prioritization of NFRs by providing constant testing and feedback, and by using techniques like MoSCoW for assessing the necessity of requirements and Delphi for estimation and forecasting, based on inputs from experts, the priorities of requirements (Unhelkar, 2018 a). Furthermore, by breaking NFRs into more manageable individual steps and including them in sprint planning as user stories, Agile teams can capture model RFRs (Corbin, 2022). “User stories are short and simple descriptions of a feature or functionality from the perspective of a user” (Jena, 2023, p. 1). In the context of Agile and NRFs, they are used to capture non-functional aspects of SD, and by using tools such as Unified Modeling Language (UML) diagrams within the Agile process, they can be modeled. See Figure 3 for an example of a UML use case diagram illustrating a user story capturing an NFR. Therefore, Agile, if used correctly, can play a significant role in gathering and modeling NFRs.

Figure 3

Use Case Diagram Capturing an NFR


Costumer Satisfaction Record System - Use Case
Alexander Ricciardi | December 21, 2024
Costumer Satisfication Record System
Services
Data Collection
Functionality Survey
<<includes>>
Ebd User
Software Engineer
Implementes, modifies, and maintains based on survey collected data survey.
Implementes, modifies, and maintains.
Example of User Story: "As an End User, I want the services to be available at all times, not just during office hours." (NFR: 24/7 availability).

Note: This is an example of a UML use case diagram illustrating a user story capturing an NFR.

Advantages and Challenges of Modeling Requirements with User Stories in an Agile Approach

As shown in Figure 3, user stories are useful for capturing and communicating requirements in Agile development. They also have other advantages as well as disadvantages that influence the modeling requirements process. They encourage collaboration between developers, customers, and stakeholders which stirs creativity and product modeling (Rehkopf, n.d.). On the other hand, it is challenging to ensure alignment and consistency of user stories across multiple teams and programs, notably the number of teams and stakeholders grows (Leanwisdom, 2024). This can lead to confusion, delays, and rework. Nonetheless, they have the advantage of simplified prioritization of work and requirements by giving estimates that make planning easier (Cohn, 2022). However, user stories can also oversimplify complex requirements if they are not carefully crafted, lack sufficient detail, or fail to capture fully the user needs and technical constraints (Choudhary, 2017). This oversimplification of complex requirements may lead to a misunderstanding of the requirements and to incomplete implementations. These advantages and disadvantages can also influence the overall design of the system, but with the right approaches and techniques, advantages can be maximized and disadvantages can be minimized.

Agile and System Design

System Design can be thought of as a solution or software design depending on the context. In Software Engineering, it can be thought of as software design, which is the process that defines the architecture, components, and interfaces of a software. The list below gives definitions of Software Engineering itself and its main components:

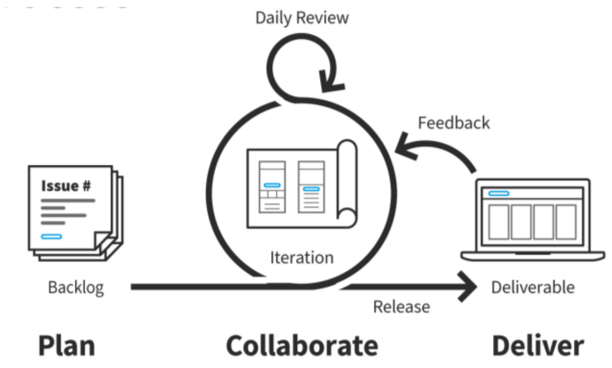
- Software Engineering (SE) is the art of engineering high-quality software solutions (Unhelkar, 2018 a).
- System Design (SyD) is part of SE, it is the process that defines the architecture, components, and interfaces of a software.

- Software Modeling (SM) is part of SyD, it is the process of project modeling based on UML to create diagrams to improve communication and participation from all project stakeholders.
- Software Development (SD) is also part of SyD, it is the process that defines activities and phases, as well as providing directions to the different teams of designers and developers throughout the Software Development Lifecycle (SDLC).

The Agile methodology's iterative, colorative, and adaptable nature can be applied to SyD. See Figure 4 for an illustration of the Agile design process as a cyclical and iterative process. First, the Agile iterating approach can be used to break down the design process into small more manageable sections. This allows the development team to collaborate with the customers or the product owner, with the goal of identifying and prioritizing the most crucial features or user stories for each specific iteration by using techniques such as user story mapping (Lteif, 2024). For example, when designing an interface, the team may start by creating a prototype to collect feedback from users and iterate on it by refining the design and then getting additional from users until the design meets the customer requirements. Secondly, the Agile colorative nature promotes teamwork between developers and stakeholders (Interaction Design Foundation, 2024). Lastly, Agile adaptable nature promotes flexibility and adaptability. In a SyD environment changes are constants. This can create challenges for teams that may struggle to adapt when priorities shift rapidly, leading to scope creep or incomplete work (Candelario, 2024). Agile can help alleviate these challenges by simplifying and clarifying priorities, creating living documentation through tools such as UML diagrams, and by implementing daily meetings to keep in focus unfinished tasks and the project design's main scope. For example, a team designing a mobile app's login screen created a prototype using UML diagrams for a simple username and password login. However, after a discussion with stakeholders based on user stories, it was decided that social media login options would improve the user's logging process. Agile adaptability nature allows the team to easily incorporate or modify this change into the existing UML diagrams during an iteration or a sprint. This discussion or meeting is called a ceremony in Agile Scrum.

Figure 4

Agile Design Process



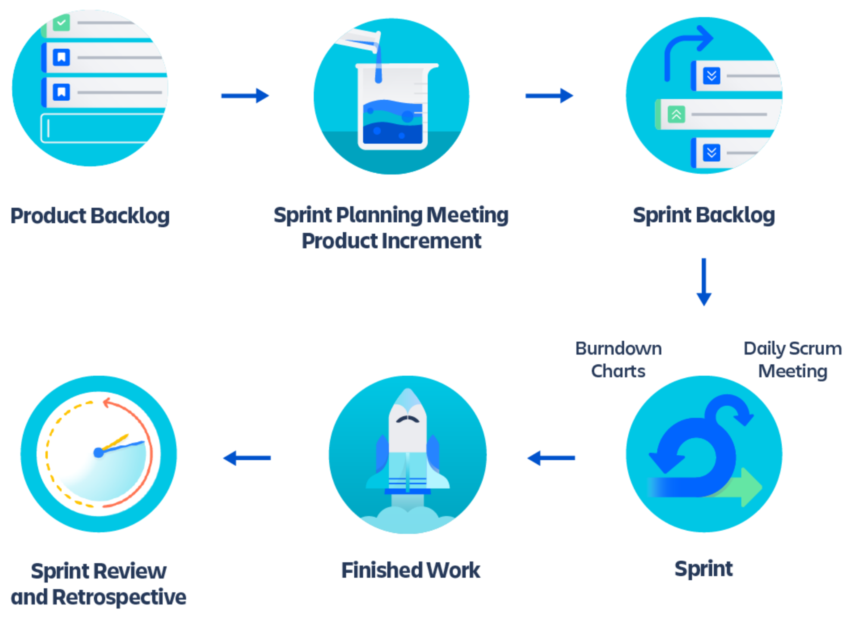
Note: this illustration was created to represent Agile design in the UX Design Processes context, but it can be applied to any system design. From "What are UX Design Processes?" by Interaction Design Foundation (2024).

Example of an Agile Role, Artifact, and Ceremony

Agile uses several elements in its methodology to design, develop, and implement software. These elements are Agile roles, artifacts, and ceremonies. Roles are responsibilities within a team rather than job titles (Chervenkova, 2022). One of those roles is defined as Product Owner. A Product Owner, in Agile Scrum teams, for example, rates the success of a project and a team's performance, it has the authority to make decisions and prioritize the Product Backlog. A Product Backlog is a prioritized list of works such as feature implementation and bug fixes. Agile artifacts are information that a team and stakeholders use to detail or describe the product being developed and the actions performed during the project (Harris, n.d.). For example, Product Backlogs are artifacts used by a team to track the work that needs to be done. The figure illustrates the main artifacts of Agile Scrum and how they are used and relate to each other in Scrum events:

Figure 5

Scrum Main Artifacts



Note: The main Agile Scrum artifacts are product backlog, sprint backlog, and increments. From “Learn about Agile Scrum Artifacts” by Harris (n.d.).

Cerneromies, also called events in Scrum, take place at the beginning of each sprint, when a team meets to discuss what their next course of action is (Laoyan, 2024). For example, A sprint planning meeting is a ceremony, it is when a team takes the time to plan out work that will be completed during the upcoming sprint. Agile elements ━roles, artifacts, and ceremonies━ provide teams with a structure that helps them to organize their work, collaborate, and track their progress.

Conclusion

Agile methodology within the realm of Software Development has an approach that prioritizes flexibility, collaboration, and customer-centricity. Within the context of Software Engineering, Agile is a project management and software development methodology that focuses on delivering smaller incremental pieces of work, rather than waiting for a single, large-scale release. Additionaly, its colorative and iterative nature allows for the continuous identification, gathering, and prioritization of NFRs. The methodology has advantages and disadvantages notably when modeling requirements with user stories, but with the right approaches and techniques advantages can be maximized and disadvantages can be minimized. In a System Design context, Agile can simplify and clarify priorities, and create living documentation through tools such as UML diagrams, and by implementing daily meetings it can keep unfinished tasks and the project design's main scope in focus. Furthermore, Agile elements like roles, artifacts, and ceremonies provide teams with a strong structure that can help them organize their work, collaborate, and track their progress. Thus, the Agile methodology is a powerful tool that helps software engineers to deliver high-quality software solutions.

References

Beckman, J. (2024, May 30). *Agile statistics: How many companies use agile in 2023? Tech report*. https://techreport.com/statistics/business-workplace/how-many-companies-use-agile/

Candelario, G., P. (2024, October 7). Agile design process: Definition, main principles, and benefits. Designrush. https://www.designrush.com/agency/ui-ux-design/trends/agile-design-process

Chervenkova, M. (2022, July 7). *Agile team roles and responsibilities: A complete guide*. Businessmap. https://businessmap.io/blog/agile-team-roles#:~:text=In%20Agile%2C%20the%20roles%20and,responsibilities%20that%20team%20members%20take.

Choudhary, N. (2017, November 13). *7 Common user story mistakes and how to avoid them?* TO THE NEW BLOG. https://www.tothenew.com/blog/7-common-user-story-mistakes-and-how-to-avoid-them/

Cohn, M. (2022, November 2). *Advantages of user stories over requirements and use cases*. Mountain Goat Software. https://www.mountaingoatsoftware.com/articles/advantages-of-user-stories-for-requirements

Corbin, M. (2022, August 1). *Catching a Cloud, Pinning it Down: How to Capture Non Functional Requirements in Agile*. Official Teamly Blog. https://www.teamly.com/blog/how-to-capture-non-functional-requirements-in-agile/

Harris, B. C. (n.d.). *Learn about Agile Scrum artifacts*. Atlassian. https://www.atlassian.com/agile/scrum/artifacts

Interaction Design Foundation. (2024, April 19). *What are UX Design Processes?*. Interaction Design Foundation - IxDF. https://www.interaction-design.org/literature/topics/ux-design-processes

Jena, S. (2023, December 5). *User stories in agile software development*. GeeksForGeeks. https://www.geeksforgeeks.org/user-stories-in-agile-software-development/

Laoyan, S. (2024, January 6). Understanding the 4 Agile ceremonies. Asana. https://asana.com/resources/agile-scrum-ceremonies

Leanwisdom. (2024, June 28). *Challenges of scaling user stories across the enterprise*. leanwisdom.com. https://www.leanwisdom.com/blog/challenges-of-scaling-user-stories-across-the-enterprise/

Lteif, G. (2024, April 10). *Part 3: Agile Design Techniques for Creating Strong and Scalable Solutions*. SoftwareDominos. https://softwaredominos.com/home/software-design-development-articles/solution-design-part-3-requirement-volatility-uncertainty-and-agile-design/

Naidu, N. (2024, March 7). Agile software development – software engineering. GeeksForGeeks. https://www.geeksforgeeks.org/software-engineering-agile-software-development/

Nguyen, M. H., Chau, T. P., Nguyen, P. X., & Bui, N. D. Q. (2024). *AgileCoder: Dynamic Collaborative Agents for Software Development based on Agile Methodology* [PDF]. https://arxiv.org/pdf/2406.11912

Mazurkiewicz, E. (2023, August 25). *Non-functional requirements - best practices*. Inwedo. https://inwedo.com/blog/non-functional-requirements-in-agile/

Paliwal, P. (2024, September 23). *What is Agile Methodology?* GeeksForGeeks. https://www.geeksforgeeks.org/what-is-agile-methodology/

Petrescu, M., & Sterca, A. (2023). *Agile Methodology in Online Learning and How It Can Improve Communication: A Case Study* [PDF]. https://doi.org/10.5220/0011317400003266

Rehkopf, B. M. (n.d.). *User stories*. Atlassian. https://www.atlassian.com/agile/project-management/user-stories

Saraev, N. (2024, June 4). *10 Types of Agile Frameworks*. As We May Think — Products & Tools for Thought. https://fibery.io/blog/product-management/agile-frameworks/

Szmit, A. (2023, September 4). *The guide to sprints in Scrum*. BigPicture. https://bigpicture.one/blog/scrum-sprints-explained/

Unhelkar, B. (2018 a). Chapter 3 - Software projects and modeling spaces: Package diagrams. *Software engineering with UML*. CRC Press. ISBN 9781138297432

Unhelkar, B. (2018 b). Chapter 1 - Software engineering fundamentals with object orientation. *Software engineering with UML*. CRC Press. ISBN 9781138297432