Agile Software Development

Chapter 3

Submitted to:

Md. Atikur Rahman

Department of Computer Science and Information Technology

Faculty of Computer Science and Engineering

Submitted by:

ID	Reg	Name
2102021	10148	Sadman Hafiz Shuvo
2102022	10149	Noshin Nazia
2102023	10150	Md. Senarul Islam
2102024	10151	Md. Sharafat Karim
2102025	10152	Seemanta Shill
2102026	10153	Nayema Ferdoushi
2102027	10154	Md. Afridi Alom Pranto
2102028	10155	Shawan Mahmud Abdullah
2102030	10157	Yasin Arafat

CONTENTS

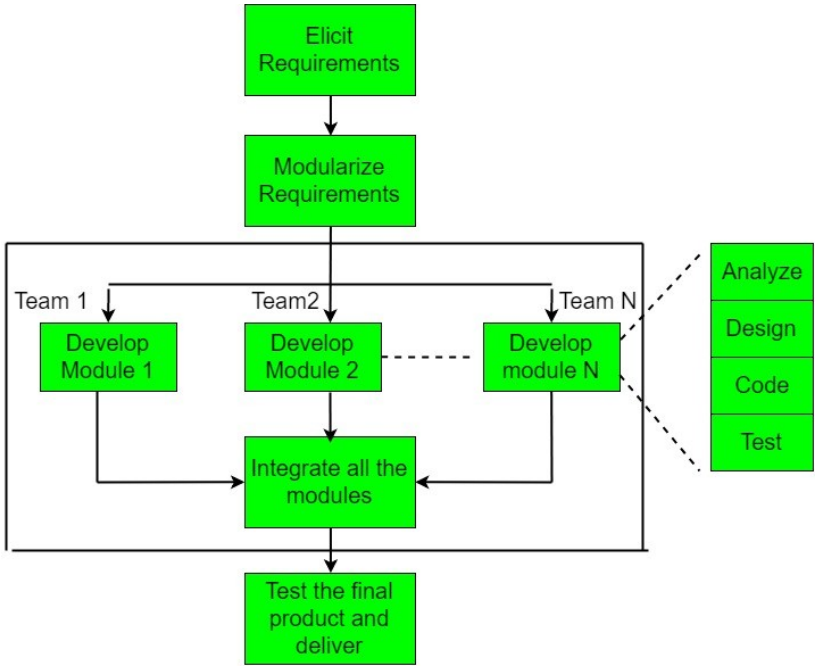
REVISION 03

Above contents have hyperlink support. Please click to directly navigate!

Agile Development

Rapid Application Development Model (RAD)

The RAD model is a type of **incremental process model in which there is an extremely short development cycle**. When the requirements are fully understood and the component-based construction approach is adopted then the RAD model is used.



The critical feature of this model is the use of powerful development tools and techniques. A software project can be implemented using this model if the project can be broken down into small modules wherein each module can be assigned independently to separate teams. These modules can finally be combined to form the final product.

Rapid software development characteristics are,

1. Specification, design and implementation are inter-leaved.
2. System is developed as a series of versions with stakeholders involved in version evaluation.
3. User interfaces are often developed using an IDE and graphical tool-set.

Agile Methods

Agile is an approach to **project management that leans heavily on short time frames, adaptability, and iteration**. It involves breaking projects into small, manageable units called iterations or sprints, which are completed in short time frames to continuously deliver functional software.

The approach prioritizes quick delivery, adapting to change, and collaboration rather than top-down management and following a set plan. In the Agile process, there is continuous feedback, allowing team members to adjust to challenges as they arise and stakeholders an opportunity to communicate consistently. Using Agile methodology, the software is distributed with fastest and fewer changes. The advantages of agile methodology are customer satisfaction by rapid, continuous development and delivery of useful software.

Agile Manifesto

The Agile Software Development Methodology Manifesto describe four core values of Agile in software development.

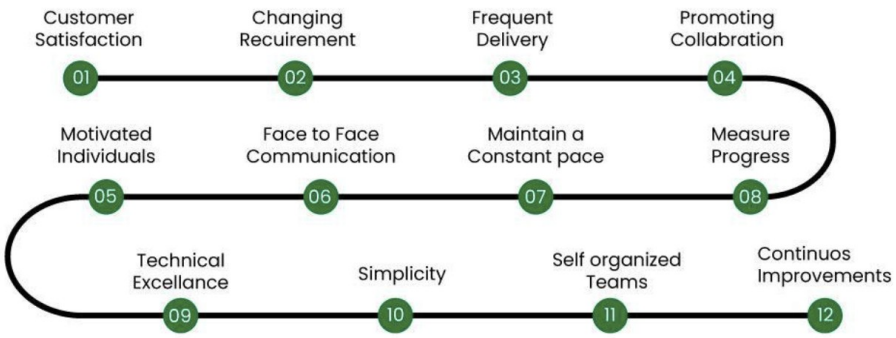
1. Individuals and Interactions over Processes and Tools
2. Working Software over Comprehensive Documentation
3. Customer Collaboration over Contract Negotiation
4. Responding to Change over Following a Plan

The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Agile Principles

These 12 core **principles** include,

1. Ensuring customer satisfaction through the early delivery of software.
2. To welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

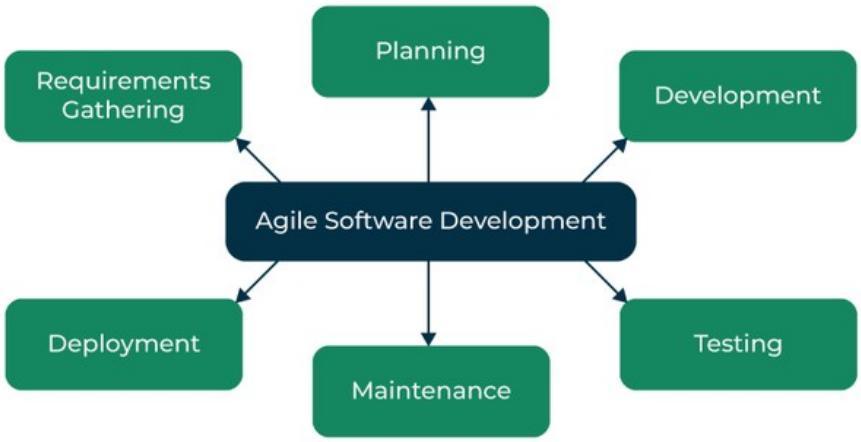


4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Keep things simple and avoid unnecessary tasks.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

The Agile Software Development Process

1. **Requirements Gathering:** The customer's requirements for the software are gathered and prioritized.
2. **Planning:** The development team creates a plan for delivering the software, including the features that will be delivered in each iteration.



3. **Development:** The development team works to build the software, using frequent and rapid iterations.
4. **Testing:** The software is thoroughly tested to ensure that it meets the customer's requirements and is of high quality.
5. **Deployment:** The software is deployed and put into use.
6. **Maintenance:** The software is maintained to ensure that it continues to meet the customer's needs and expectations.

Advantages of Agile

- Deployment of software is quicker and thus helps in increasing the trust of the customer.
- It can better adapt to rapidly changing requirements and respond faster.
- Helps in getting immediate feedback which can be used to improve the software in the next increment.
- People – Not Process. People and interactions are given a higher priority than processes and tools.
- It emphasizes collaboration and communication among team members, stakeholders, and customers. This leads to improved understanding, better alignment, and increased buy-in from everyone involved.
- It prioritizes customer satisfaction and focus on delivering value to the customer.
- Agile methodologies promote a collaborative, supportive, and positive work environment.

Disadvantages of Agile

- In the case of large software projects, it is difficult to assess the effort required at the initial stages of the software development life cycle.
- Agile Development is more code-focused and produces less documentation.
- Agile development is heavily dependent on the inputs of the customer. If the customer has ambiguity in his vision of the outcome, it is highly likely that the project to get off track.
- Face-to-face communication is harder in large-scale organizations.
- Only senior programmers are capable of making the kind of decisions required during the development process. Hence, it's a difficult situation for new programmers to adapt to the environment.
- **Lack of predictability:** Agile Development relies heavily on customer feedback and continuous iteration, which can make it difficult to predict project outcomes, timelines, and budgets.
- **Lack of emphasis on testing:** Agile Development places a greater emphasis on delivering working code quickly, which can lead to a lack of focus on testing and quality assurance. This can result in bugs and other issues that may go undetected until later stages of the project.
- **Risk of team burnout:** Agile Development can be intense and fast-paced, with frequent sprints and deadlines. This can put a lot of pressure on team members.

Agile method applicability

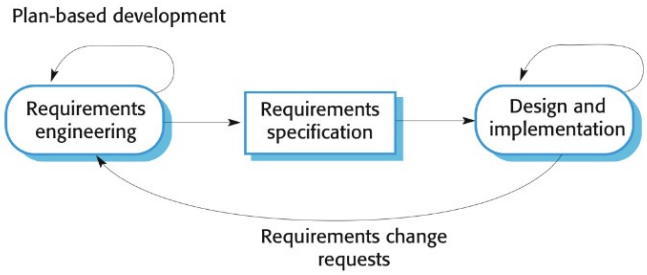
Some agile method is applicable in such conditions,

1. Product development where a software company is developing a small or medium-sized product for sale
2. Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are not a lot of external rules and regulations that affect the software.

Because of their focus on small, tightly-integrated teams, there are problems in scaling agile methods to large systems.

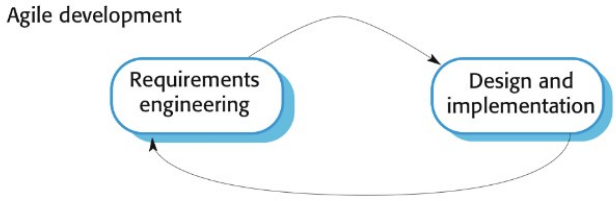
Plan-driven and Agile Development

A plan-driven approach to software engineering is a traditional, linear method of software development. It is based on a sequential, phase-by-phase approach, where each phase has a specific set of activities, deliverables, and timelines. The entire development process is planned and scheduled in advance, with clear goals and objectives defined for each stage.

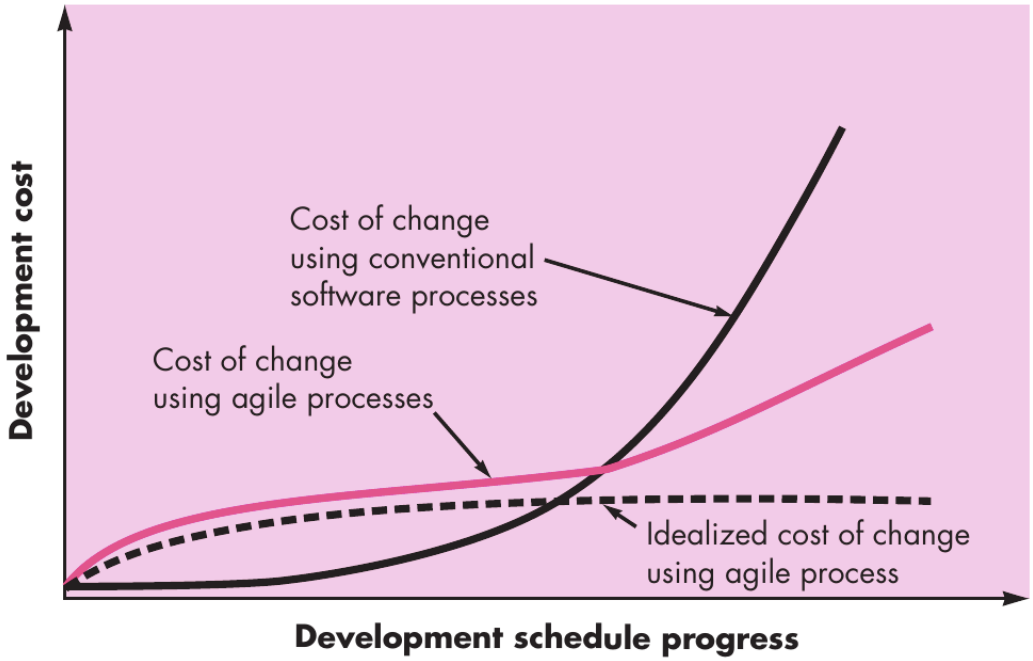


Each stage is completed before moving on to the next one, and the outputs from each stage are well-defined and documented. The plan-driven approach relies heavily on upfront planning, and changes to the plan are often difficult and costly to implement.

Agile development, on the other hand, is an iterative and incremental approach to software development. It emphasizes flexibility, collaboration, and rapid delivery. In agile development, the traditional stages of software development (specification, design, implementation, and testing) are inter-leaved, and the outputs from the development process are decided through a process of negotiation and collaboration among stakeholders.



Besides, Plan-driven development, such as the Waterfall model, typically involves higher upfront costs due to extensive planning, requirements gathering, and documentation before coding begins. This approach can be cost-effective for projects with stable, well-defined requirements but can become expensive if changes are needed later in the process, as it lacks flexibility. The linear development process also means that errors discovered late in the project can lead to costly rework.



In contrast, Agile development usually has lower upfront costs and allows for continuous feedback and iterative development, making it more adaptable to change. This flexibility often results in lower overall costs, especially in dynamic environments where requirements evolve over time. Agile's focus on delivering small, incremental value can reduce the risk of extensive rework, potentially lowering long-term maintenance costs and improving the return on investment.

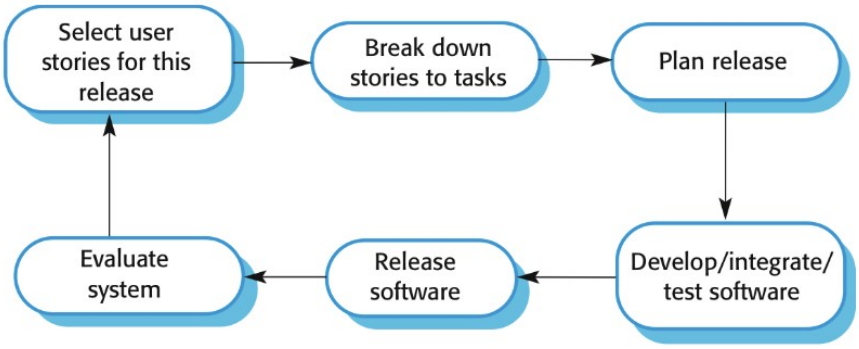
Extreme programming (XP)

Extreme Programming (XP) is an **Agile software development methodology that focuses on delivering high-quality software through frequent and continuous feedback, collaboration, and adaptation**. XP emphasizes a close working relationship between the development team, the customer, and stakeholders, with an emphasis on rapid, iterative development and deployment.

Good Practices in Extreme Programming

Some of the good practices that have been recognized in the extreme programming model to maximize their use are given below:

- **Code Review:** Code review detects and corrects errors efficiently. It suggests pair programming as coding and reviewing of written code carried out by a pair of programmers who switch their work between them every hour.
- **Testing:** Testing code helps to remove errors and improves its reliability. XP suggests test-driven development (TDD) to continually write and execute test cases. In the TDD approach, test cases are written even before any code is written.
- **Incremental development:** Incremental development is very good because customer feedback is gained and based on this development team comes up with new increments every few days after each iteration.



- **Simplicity:** Simplicity makes it easier to develop good-quality code as well as to test and debug it.
- **Design:** Good quality design is important to develop good quality software. So, everybody should design daily.
- **Integration testing:** Integration Testing helps to identify bugs at the interfaces of different functionalities. Extreme programming suggests that the developers should achieve continuous integration by building and performing integration testing several times a day.

XP and agile principles

XP is closely aligned with the core principles of Agile methodologies, including,

- **Customer Satisfaction:** XP prioritizes customer involvement and feedback to ensure that the product meets their needs.
- **Embrace Change:** XP welcomes changes throughout the development process, adapting to evolving requirements as needed.
- **Deliver Working Software Frequently:** XP emphasizes small, frequent releases to provide value to customers early on.
- **Collaboration:** XP promotes teamwork and collaboration between developers, customers, and stakeholders.

- **Continuous Improvement:** XP encourages regular refactoring and process improvements to enhance the quality of the software and the development process.

Advantages of Extreme Programming (XP)

- The main advantage of Extreme Programming is that this methodology allows software development companies to save costs and time required for project realization. Time savings are available because, XP focuses on the timely delivery of final products.
- Extreme Programming teams save lots of money because they don't use too much documentation. They usually solve problems through discussions inside the team.
- Simplicity is one more advantage of Extreme Programming projects. The developers who prefer to use this methodology create extremely simple code that can be improved at any moment.
- The whole process in XP is visible and accountable. Developers commit what they will accomplish and show progress.
- Constant feedback is also the strong side. It is necessary to listen and make any changes needed in time.
- XP assists to create software faster thanks to the regular testing at the development stage.
- Extreme Programming contributes increasing employee satisfaction and retention.

Refactoring

Code refactoring in Extreme Programming (XP) **is the process of improving the internal structure of existing code without changing its external behavior**.

Most of the modern software developers are used to keeping their code intact after they write it. They do not want to change anything until it works. The old code may have some bugs or other flaws, but they will still use it because they don't want to work on its improvement. In Extreme Programming things go differently. It is an Agile method and so flexibility is its main feature.

Refactoring is a practice of software development that allows you to improve the code without changing or breaking its functionality. It is aimed at simplification. The code should be as simple as possible to find all the bugs and to make it easy to change something in the process of project's performance.

Testing in XP

Testing is a core component of Extreme Programming (XP), and it is integrated into every step of the development process. XP emphasizes continuous testing to ensure the quality and reliability of the software. The testing practices in XP are designed to catch issues early, promote confidence in the code, and support frequent refactoring.

Pair programming

Pair programming is a software development practice where two programmers work together on one computer. It involves one programmer, the driver, writing code while the other, the observer or navigator, reviews each line of code as it's typed. This real-time collaboration helps catch errors early, improves code quality through constant feedback, and ensures better design decisions. It also facilitates knowledge sharing and reduces the likelihood of bugs, leading to more efficient problem-solving and enhanced productivity within the development team.

Advantages of Pair Programming

Given below are some advantages of pair programming:

- **Two brains are always better than one:** If driver encounters a problem with code, there will be two of them who'll solve problem. When driver is writing code, navigator can think about a solution to problem.
- **Detection of coding mistakes becomes easier:** Navigator is observing each and every line of code written by driver, so mistakes or error can be detected easily.
- **Mutual learning:** Both of them can share their knowledge with each other and can learn many new things together.
- **Team develops better communication skills:** Both of them share knowledge and work together for many hours a day and constantly share information with each other, so this can help in developing better communication skills, especially when one of members is a newbie and other is an expert.
- **Better Designs**: Two perspectives mean better ideas. The navigator can think about the big picture while the driver focuses on the details, resulting in more thoughtful and robust designs.
- **Improved Focus**: It's easier to stay on track. Pairing helps maintain concentration because both programmers are actively involved, reducing distractions and procrastination.
- **Team Bonding**: Working closely builds stronger teams. Pair programming fosters communication and trust, creating a positive work environment where team members support each other.
- **Higher Quality Code**: Collaboration leads to better solutions. By discussing and debating options in real-time, pair programming produces cleaner, more maintainable code that meets the project's requirements.

Scrum

Scrum is a framework for project management that emphasizes **teamwork, accountability and iterative progress** toward a well-defined goal. **It is a way of managing projects**, especially in software development.

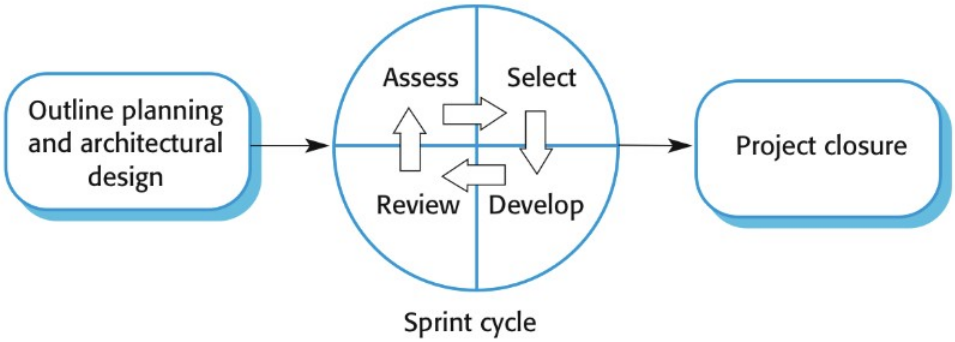
It’s like a playbook that teams use to work together more effectively. Instead of doing everything at once, Scrum breaks work into smaller parts called “sprints.” Each sprint focuses on completing a specific piece of the project, allowing teams to adapt and improve as they go. It’s all about teamwork, communication, and getting things done step by step.

Three phases in Scrum include,

1. An outline planning for the project and design the software architecture.
2. A series of sprint cycles, where each cycle develops an increment of the system.
3. The project closure phase wraps up the project, completes required documentation.

The Scrum Processes

A Scrum sprint cycle is a time-boxed period when a team delivers a set amount of work. It includes task planning, performing, managing the tasks, attending daily stand-ups, and communicating with the Scrum teams. The outcome of Sprint Execution is a potentially



shippable product increment, formed from a list of product backlog items by meeting the team members' definition of done.

Teamwork in Scrum

A scrum team is a group of collaborators, typically between five and nine individuals, who work toward completing projects and delivering products. The fundamental scrum team comprises one scrum master, one product owner and a group of developers. Within a scrum team, there is no rank or hierarchy.

The scrum methodology emphasizes teamwork in project management. It stresses accountability and iterative progress toward a well-defined goal. Scrum is part of agile software development and teams practice agile.

Everyone on the Scrum Team is accountable for certain aspects of the team’s work. The Scrum Team as a whole is accountable for creating a valuable, useful Increment every Sprint and the Product Owner, Developers and Scrum Masters each have accountabilities that are specific to them.

Advantages and Disadvantages

Advantage	Disadvantage
1. Motivated to finish the sprint.	**1.** Losing the track of the process.
2. All team-members have the access to project progress.	**2.** Not clear defining each role.
3. The focus on quality.	**3.** No big picture of the project.
4. Easy to recognize.	

Large Systems Development

Large systems are usually collections of separate, communicating systems, where separate teams develop each system. Frequently, these teams are working in different places, sometimes in different time zones.

Large systems have a long procurement and development time. It is difficult to maintain coherent teams who know about the system over that period as, inevitably, people move on to other jobs and projects.

Scaling up agile methods

Scaling up is concerned with using agile methods for developing large software systems that cannot be developed by a small team .

Scaling out is concerned with how agile methods can be introduced across a large organization with many years of software development experience.

The trend of scaling Agile to address the needs of large enterprises and complex projects is expected to gain momentum. Organizations are increasingly adopting Agile Scaling frameworks like SAFe, LeSS, and Nexus to coordinate multiple teams, align strategic initiatives, and achieve seamless collaboration across the enterprise.

Scaling up to large systems

For large systems development, it is not possible to focus only on the code of the system. **Scaling up** refers to the vertical expansion of Agile practices to manage complexity at higher organizational levels. It focuses on increasing the scope of Agile implementation beyond individual teams to include programs, portfolios, and the overall enterprise.

Scaling out to large companies

Scaling out refers to the horizontal expansion of Agile practices across multiple teams working in parallel on different parts of a large system. It involves ensuring that these teams can work effectively together while maintaining Agile principles.

References

1. **Software Engineering Book : A Practitioner's Approach***
   (by Roger S. Pressman)
2. https://en.wikipedia.org/wiki/Agile_software_development
3. https://www.coursera.org/articles/what-is-agile-a-beginners-guide
4. https://www.geeksforgeeks.org/software-engineering-rapid-application-development-model-rad/
5. https://www.javatpoint.com/agile
6. https://agilemanifesto.org/principles.html
7. https://www.freecodecamp.org/news/agile-software-development-handbook/
8. https://www.geeksforgeeks.org/software-engineering-extreme-programming-xp/
9. https://hygger.io/blog/disadvantages-and-advantages-of-extreme-programming/#advantages-of-extreme-programming
10. https://hygger.io/blog/refactoring-in-extreme-programming/
11. https://www.scrum.org/resources/what-scrum-team
12. https://www.geeksforgeeks.org/scrum-software-development/