Extreme Programming in Agile systems development

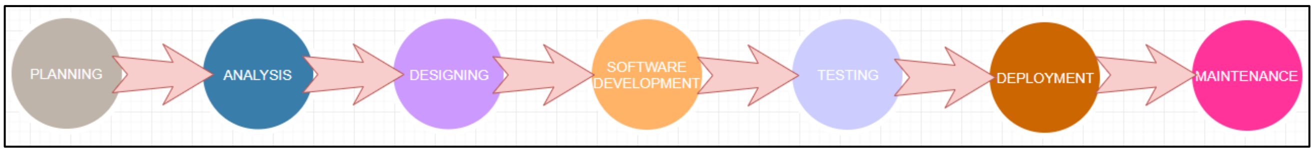
Leandro Maglianella (K1833503)

Kingston University

Introduction

The modern world owes its existence to the scientific revolutions that made the development of software and technological systems possible. A system to be implemented and work properly and effectively needs developers who follow a precise working model. In fact, there are many possible existing approaches and methods, called System Development Life Cycles (SDLC), each characterised by its own advantages and disadvantages. It is essential that developers know them so that they can always choose the one most suitable to the system to be developed, in order not to slow down its progress.

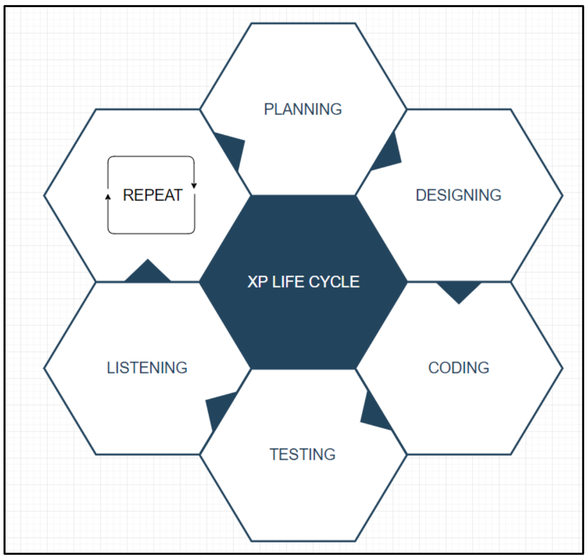
SDLCs describe each action taken in the software development process step by step and are generally divided into phases:



Although each SDLC shares these same steps with the other SDLCs, in each of them they are implemented in different ways. Currently, one of the most popular methodology is Agile, which bases its strength on simplicity, speed and flexibility: the development is divided into small periods of 2-4 weeks called "sprints", when a sprint terminates the product obtained is released to the customer for feedback, which will form the basis of the requirements of the next sprint. By doing this, it is possible to add specifications during development in an efficient way and usually without great costs, increasing customer satisfaction. All this is not possible using a traditional software development methodology like Waterfall, in which every software requirement must be clearly prearranged.

We will focus particularly on the Extreme Programming (XP) method, one of the available Agile approaches. The origins of XP can be traced back to Kent Beck who created it in 1996 and published a book in 1999 where he describes the values and principles to be respected in his methodology and derives a list of practices and behaviours to be applied in software development. In this article all these concepts will be considered and discussed in detail.

XP Life cycle stages



1. Planning

This is the first phase of the Extreme Programming life cycle: the customer meets the development team, provides it with the "user stories" and the requirements of the software to be produced are established. The team uses user stories to derive the iterations to follow in the development and establishes the time, costs and a plan to complete them.

2. Designing

The team design how the code should be constructed respecting the plan. Metaphors and common standards are established and ideas are sought to implement specific functionalities. Every aspect must be designed in the simplest way possible and in such a way that the work of each team member is compatible with the work of the other members.

3. Coding

This is the main phase in the XP life cycle, in fact XP is characterized by giving greater importance to programming instead of documentation to quickly provide results to the customer.

4. Testing

This phase is performed during coding, several unit tests are performed to ensure that the code is free of bugs before release and that it passes the customer's acceptance tests.

5. Listening

The cycle ends with the feedback from the customer and the project manager regarding the software developed, which will become the foundation of a new design. After this phase, the cycle is repeated until the customer is satisfied.

Project roles

Customer

The person who commissions the project, establishes the specifications of the software to be developed via user stories and usually he is the one who pays it, other times there is an external funder called “gold owner”.

Programmer

The person who receives the user stories and extracts tasks from them, writes the code and runs unit tests on it.

Coach

The person who checks if the project is following the established plan.

Tracker

The person who monitors the progress and helps with the coding, if necessary sets up meetings with the customer.

Tester

The person who runs tests on the software to ensure that it adequately meets the customer's requirements.

Doomsayer

The person who reports any problems in the code that could later lead to major errors.

Manager

The person who ensures that the various meetings take place as and when they are scheduled.

Values

1. Communication

Usually in XP the code is not documented, it is necessary that the team members communicate with each other and with the customer.

2. Simplicity

The requirements of the software must be achieved with simplicity, complex solutions must be avoided in order to save time and effort and so that the code can be easily modified later. Only the requirements imposed by the customer must be addressed.

3. Feedback

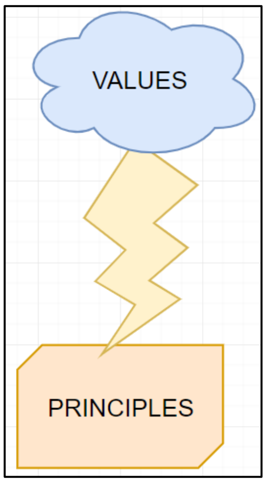
In XP, feedback (being at the base of every development cycle) plays a very important role. They can come from the customer or from any team member, usually from the tester.

4. Courage

This value concerns the ability to objectively evaluate the work done, to inform the team of any problems, to be always ready to face changes and innovations in the project and to accept feedback constructively.

5. Respect

It is a priority that the team members establish a peaceful relationship, that they respect each other and respect the customer, respect the specifications and the work done by all towards the common goal.



Principles

Let us now describe the 5 principles derived from the values:

1. Rapid feedback

Team members must have the ability to effectively and quickly respond to feedback.

2. Assumed simplicity

Work should focus only on the necessary tasks, developers should not repeat themselves and should not focus on aspects that could be useful later.

3. Incremental changes

The project must be developed gradually little by little, it should not receive new big features in a single update.

4. Embracing changes

The team must welcome and positively accept all customer opinions that lead to having to modify the software and implement them.

5. Quality work

The team must engage in work and produce high quality software.

The importance of capturing the "story"

User stories are simple short sentences usually written using "story cards" that describe how the various components of the software should relate to each other and how the software is supposed to behave in general. As it is logical to imagine, user stories are of great importance: they ideally represent the software and are the specifications to follow to create the product desired by the customer. It is not generally easy for the customer to imagine how he would like the end result to be as it will often be complex and articulated, thanks to the stories it is the developer himself who has to guess what the client's perspective is and correctly frame the best path to follow to satisfy him.

Activities

1. Designing

Usually this activity, considered the main one in the other development methodologies, in the Extreme programming must be kept short, simple and direct to allow the writing of equally slender code.

2. Coding

It is the core of XP, only after having a basic software it will be possible to improve it according to the customer's indications.

3. Testing

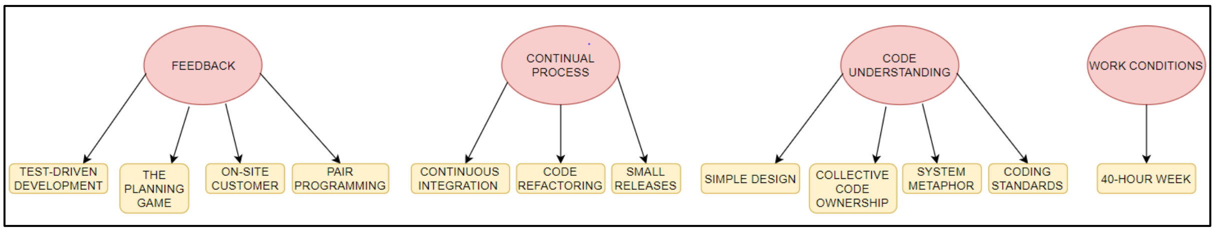
The software is incrementally built, every function must be rigorously tested by means of unit tests and acceptance tests before the programmer begins to face the next feature.

4. Listening

This activity expresses the importance of the ability to know how to interpret the desires of the customer by the programmer, to satisfy him in the best and simplest of way.

Practices

Finally, the following 12 practices were extracted from the values and activities, which can be grouped into 4 groups:

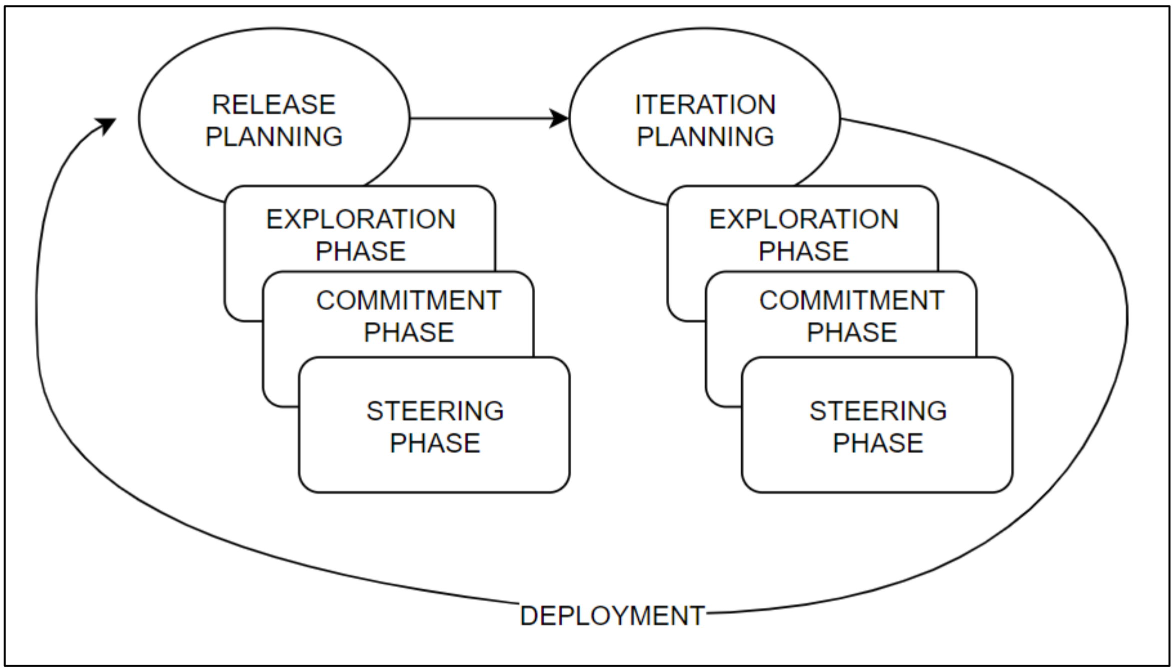


1. Test-driven development

This practice consists of developing tests before designing the code itself, so that the programmer has the opportunity to reflect on which problems could cause bugs before coming across them and to approach the implementation of new features more consciously. It aims to minimize bugs.

2. The planning game

This practice describes the planning process to be followed in software development, usually each iteration lasts a few weeks:



2.1 Release planning

- *Exploration phase:* The customer defines the requirements for the project through the use of user story cards. Each user story is analysed and those that are complex are split into multiple user stories.
- *Commitment phase:* For each user story are estimated the costs, benefits and time to implement and are classified according to priority. At this point the team decides which user stories to implement in the current iteration.

- *Steering phase:* In this phase the team has the possibility to make changes to the plan before starting the implementation.

2.2 Iteration planning

- *Exploration phase:* This phase consists of converting previously selected user stories into tasks (written on task cards) to be performed. Each task is analysed and those that are complex are split into multiple tasks, those that are too small are combined in a bigger task.
- *Commitment phase:* Tasks are allocated to each programmer of the team, who estimate the load factor, which is the time required to complete the tasks. Using these estimations the tasks are redistributed and balanced so that each programmer has a similar amount of work.
- *Steering phase:* Programmers get together in pairs, design and implement the features of their tasks and run tests. After being fully tested, the software is ready to be deployed.

3. On-site customer

The XP philosophy states that the customer (or his representative) must actively work on the development of the software and must always be available and present with the team to answer their questions and establish which issues should be addressed first.

4. Pair Programming

The code is developed by two programmers who work together on the same computer: the first focuses on writing the code while the second checks its correctness and suggests improvements, also in relation to what was developed previously. After a period of time the programmers switch roles and also exchange between the other pairs to get a good knowledge of the whole project. This practice slows down the coding process but the code developed is usually of the highest quality.

5. Continuous integration

To emphasize the communication value, it is essential that every programmer works locally on the most recent version of the code developed to limit the occurrence of integration problems. The written code is uploaded and added to the collective project every few hours to be available to the whole team.

6. Code refactoring

The software must always maintain its simplicity. The programmer must take care of the code making it consistent, removing unnecessary, duplicate or redundant functions.

7. Small releases

This practice consists in deploying the first version of the software as soon as possible and then improving it by adding new features step by step. In this way it is possible to receive feedback more frequently, allow the customer to gain confidence by interacting with the software and observe how the software really works.

8. Simple design

As for "code refactoring", this practice also concerns the simplicity of the project. The code must contain the least number of classes and methods and must be able to produce exactly what the programmer desire. Only one feature at a time must be implemented.

9. Collective code ownership

Every programmer should feel responsible for the entire code, if he finds a bug he should feel free and encouraged to fix it immediately. The programmer must be familiar with the entire code, including what the other programmers have written.

10. System metaphor

The software must be designed in such a way that it can be understood by everyone. Variables or classes or methods must be declared with clear names that easily imply their purpose.

11. Coding standards

The team must agree on what style and format the code should have and what programming techniques should be avoided and which ones should be used. This increases the readability of the code and promotes collective ownership.

12. 40-hour week

This last practice declares that a programmer is able to work best only if well rested. To maintain creativity and abstraction skills at a high level, the work week should last at most 40 hours.

Conclusion

XP is clearly one of the best and most articulated development methodologies and all its aspects have been discussed. Obviously it is not always applicable because it requires that the entire team has great coordination and experience, without which it would be almost impossible to complete a project. However, it is perfect for small teams (for example in a start-up project) that are committed to following its values, principles and practices.

References

- Diagrams made with: https://www.draw.io/

- 8 Models of the System Development Life Cycle. (2017) Available at: https://www.innovativearchitects.com/KnowledgeCenter/basic-IT-systems/8-SDLC-models.aspx (Accessed: 8 November 2019).

- Azimi, A. (2017) Extreme Programming (XP). Available at: https://medium.com/@HazzAzimi/extreme-programming-xp-35223784976e (Accessed: 11 November 2019).

- Extreme Programming. (2019) Available at: https://www.agilealliance.org/glossary/xp/#q=~(infinite~false~filters~(postType~(~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'xp))~searchTerm~'~sort~false~sortDirection~'asc~page~1) (Accessed: 11 November 2019).

- Extreme Programming: Values, Principles, and Practices. (2018) Available at: https://www.altexsoft.com/blog/business/extreme-programming-values-principles-and-practices/ (Accessed: 11 November 2019).

- Franklin, A. (2019) Top 6 SDLC Methodologies And How To Choose The Best One? Available at: https://www.goodcore.co.uk/blog/sdlc-methodologies/ (Accessed: 8 November 2019).

- Hutagalung, W. (2006) Extreme Programming. Available at: http://www.umsl.edu/~sauterv/analysis/f06Papers/Hutagalung/#xp (Accessed: 18 November 2019).

- Rasmusson, J. (no date) Extreme Programming. Available at: http://www.agilenutshell.com/xp (Accessed: 11 November 2019).

- Singh, V. (2019) Top 7 SDLC Methodologies. Available at: https://hackr.io/blog/sdlc-methodologies (Accessed: 8 November 2019).

- Synopsys Editorial Team, (2017) Top 4 software development methodologies. Available at: https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/ (Accessed: 8 November 2019).

- The 5 Important Values Of Extreme Programming (Powerful). (2018) Available at: https://www.educba.com/all-about-extreme-programming/ (Accessed: 11 November 2019).

- Wikipedia. (2019) Extreme programming practises. Available at: https://en.wikipedia.org/wiki/Extreme_programming_practices (Accessed: 18 November 2019).