Extreme Programming: Values, Principles, and Practices

With software engineering existing in such a fast-paced environment, traditional project management approaches are no longer viable. That means that IT professionals must find new ways to handle frequently changing development tasks.

Sharing this idea and focusing on the existing incremental development techniques, 17 software specialists introduced the Agile project management philosophy in 2001. Principles of flexible, fast, and collaboration-centered software development were outlined in the Agile Manifesto.

Extreme Programming (XP) is one of the numerous Agile frameworks applied by IT companies. But its key feature — emphasis on technical aspects of software development — distinguishes XP from the other approaches.

Software engineer Ken Beck introduced XP in the 90s with the goal of finding ways to write high-qualitative software quickly and being able to adapt to customers' changing requirements. In 1999, he refined XP approaches in the book *Extreme Programming Explained: Embrace Change*.

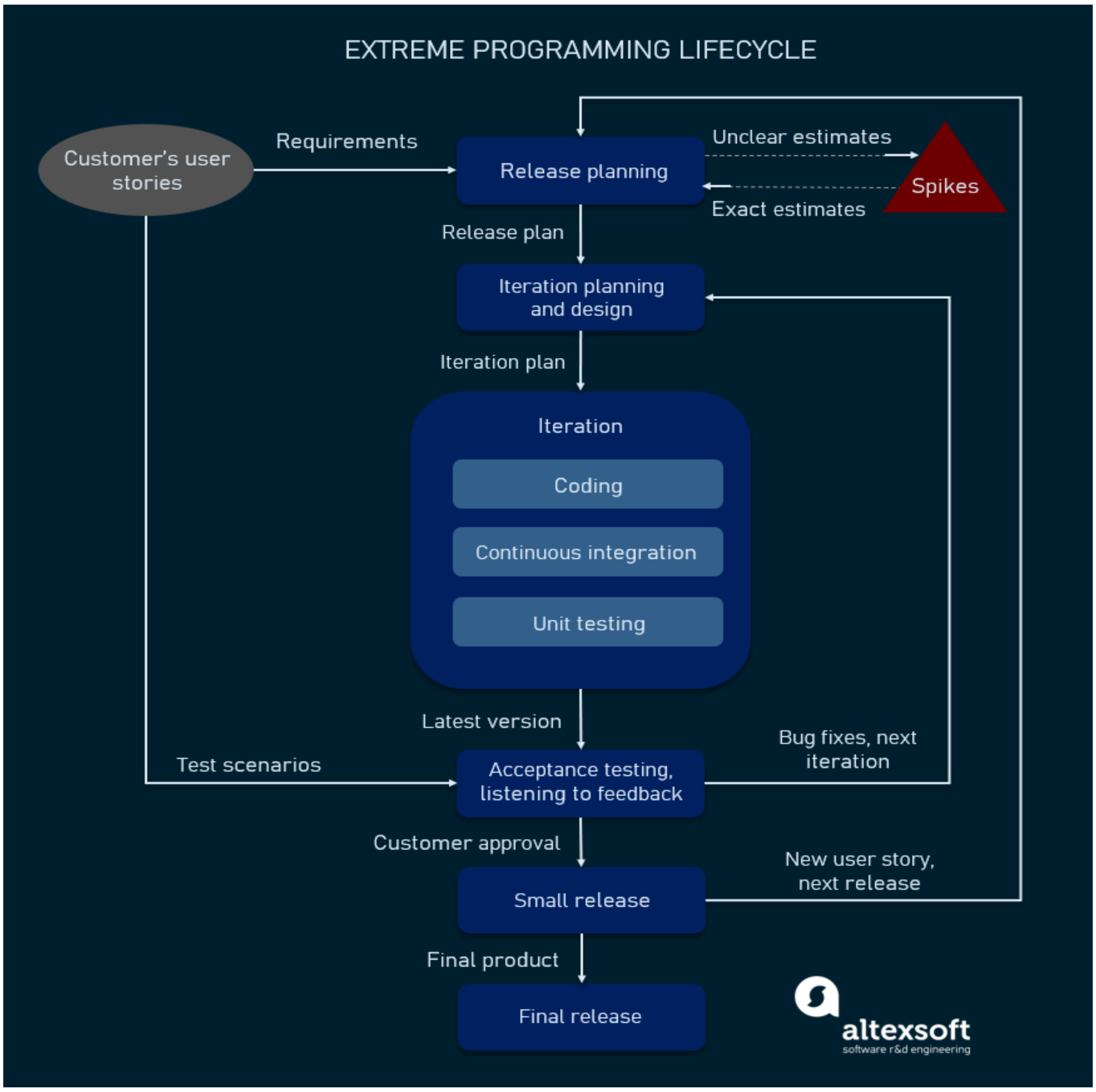
XP is a set of engineering practices. **Developers have to go beyond their capabilities while performing these practices**. That's where the "extreme" in the framework's title comes from. To get a better understanding of these practices, we'll start with describing XP's lifecycle and the roles engaged in the process.

The process and roles of extreme programming

The XP framework normally involves 5 phases or stages of the development process that iterate continuously:

1. **Planning,** the first stage, is when the **customer meets the development team and presents the requirements in the form of user stories to describe the desired result**. The team then estimates the stories and creates a release plan broken down into iterations needed to cover the required functionality part after part. If one or more of the stories can't be estimated, so-called *spikes* can be introduced which means that further research is needed.

2. **Designing** is actually a part of the planning process, but can be set apart to emphasize its importance. It’s related to one of the main XP values that we’ll discuss below — **simplicity**. A good design brings logic and structure to the system and allows to avoid unnecessary complexities and redundancies.
3. **Coding** is the phase during which the actual code is created by implementing specific XP practices such as **coding standards**, pair programming, continuous integration, and collective code ownership (the entire list is described below).
4. **Testing** is the core of extreme programming. It is the regular activity that involves both unit tests (automated testing to determine if the developed feature works properly) and acceptance tests (customer testing to verify that the overall system is created according to the initial requirements).
5. **Listening** is all about constant communication and feedback. The customers and project managers are involved to describe the business logic and value that is expected.



XP lifecycle in a nutshell

Such a development process entails the cooperation between several participants, each having his or her own tasks and responsibilities. Extreme programming puts people in the center of the system, emphasizing the value and importance of such social skills as communication, cooperation, responsiveness, and feedback. So, these roles are commonly associated with XP:

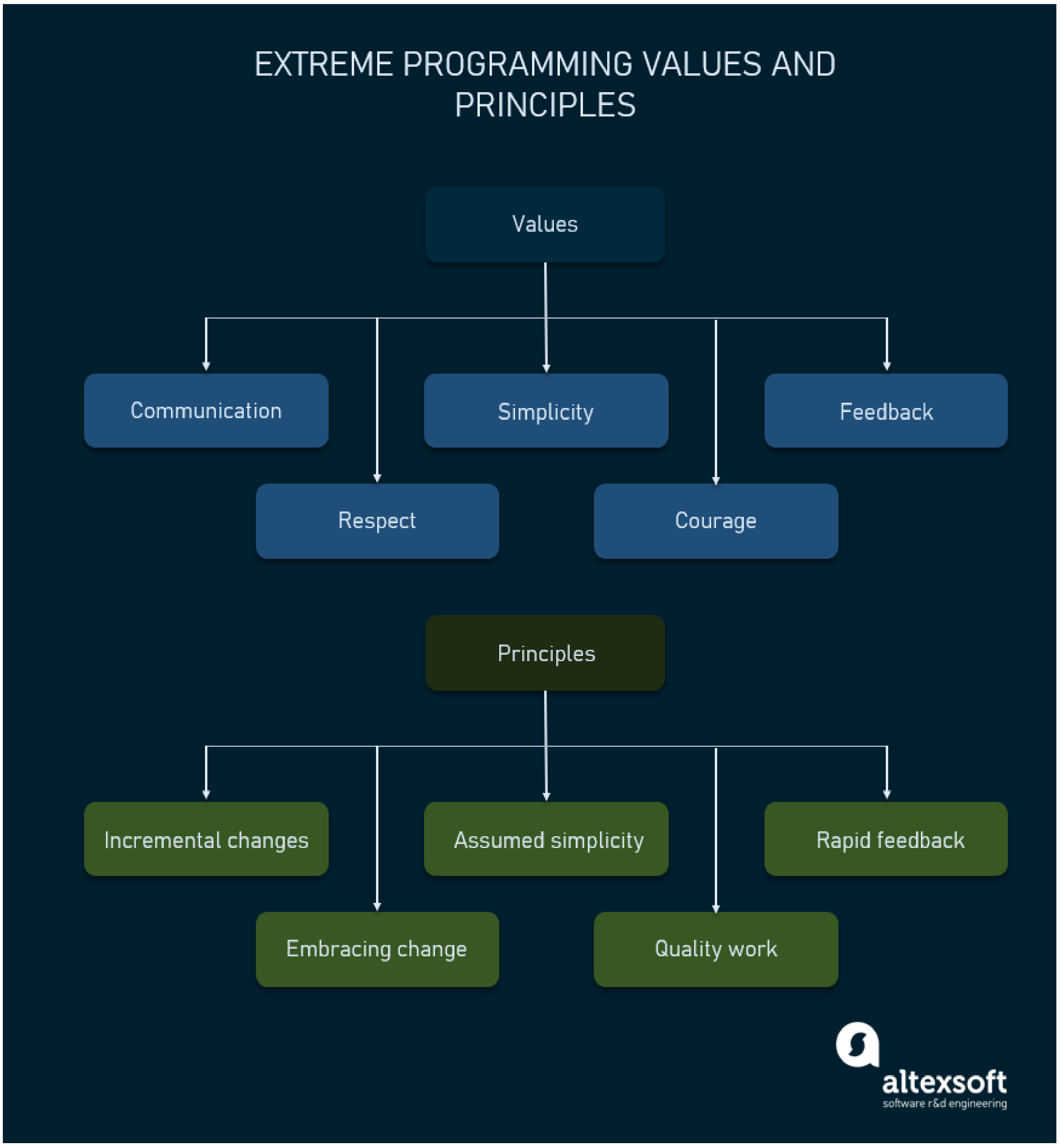
1. **Customers** are expected to be heavily engaged in the development process by creating user stories, providing continuous feedback, and making all the necessary business decisions related to the project.
2. **Programmers or developers** are the team members that actually create the product. They are responsible for implementing user stories and conducting user tests (sometimes a separate **Tester** role is set apart). Since

XP is usually associated with cross-functional teams, the skill set of such members can be different.

3. **Trackers or managers** link customers and developers. It’s not a required role and can be performed by one of the developers. These people organize the meetups, regulate discussions, and keep track of important progress KPIs.
4. **Coaches** can be included in the teams as mentors to help with understanding the XP practices. It’s usually an outside assistant or external consultant who is not involved in the development process, but has used XP before and so can help avoid mistakes.

Values and principles of extreme programming

In the late 90s, Ken Beck summarized a set of certain values and principles that describe extreme programming and lead to more effective cooperation within the team and, ultimately, higher product quality.



Values and principles of XP

Values of extreme programming

XP has simple rules that are based on 5 values to guide the teamwork:

1. **Communication.** Everyone on a team works jointly at every stage of the project.

2. **Simplicity.** Developers strive to write simple code bringing more value to a product, as it saves time and effort.
3. **Feedback.** Team members deliver software frequently, get feedback about it, and improve a product according to the new requirements.
4. **Respect.** Every person assigned to a project contributes to a common goal.
5. **Courage.** Programmers objectively evaluate their own results without making excuses and are always ready to respond to changes.

These values represent a specific mindset of motivated team players who do their best on the way to achieving a common goal. XP principles derive from these values and reflect them in more concrete ways.

Principles of extreme programming

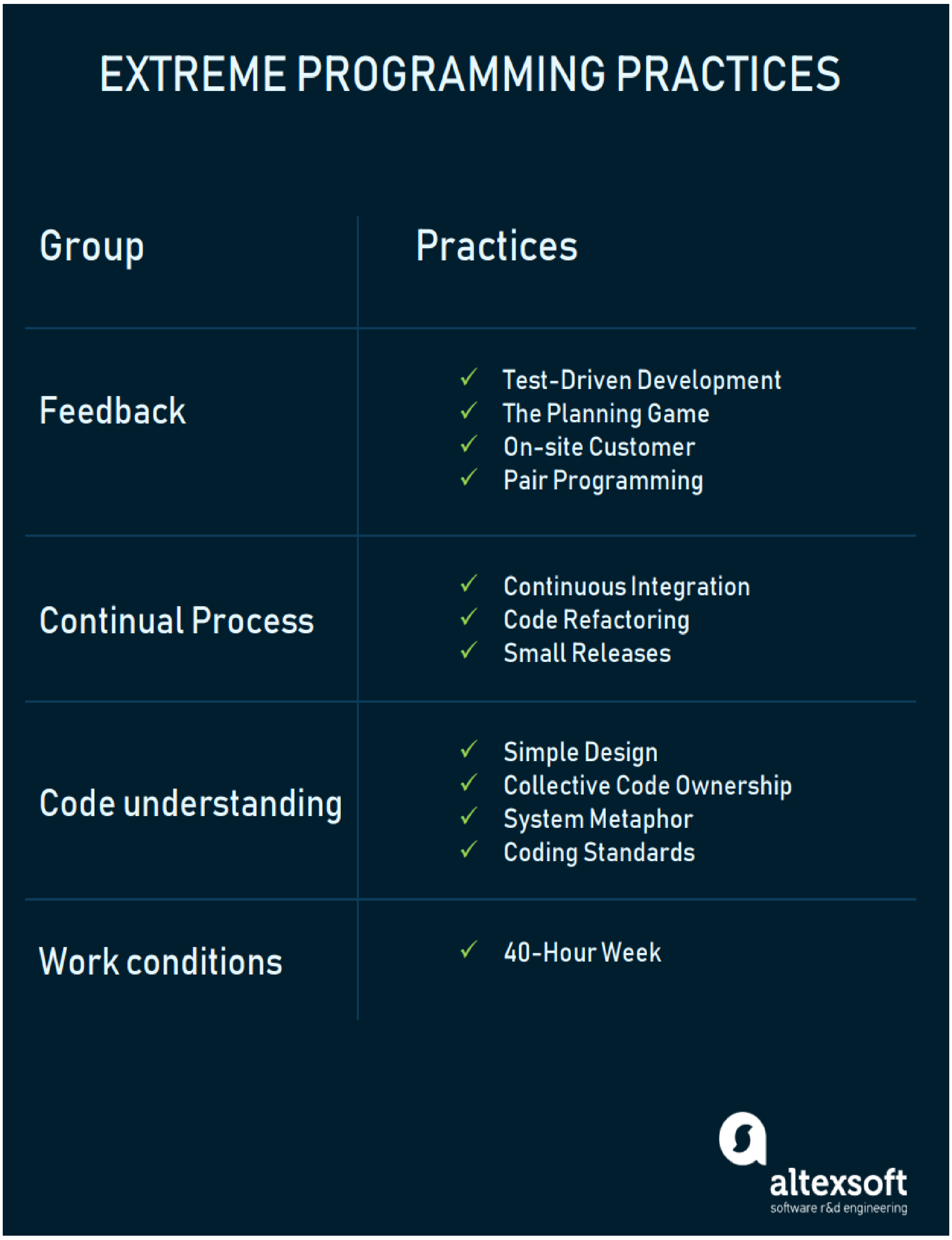
Most researchers denote 5 XP principles as:

1. **Rapid feedback.** Team members understand the given feedback and react to it right away.
2. **Assumed simplicity.** Developers need to focus on the job that is important at the moment and follow YAGNI (You Ain't Gonna Need It) and DRY (Don't Repeat Yourself) principles.
3. **Incremental changes.** Small changes made to a product step by step work better than big ones made at once.
4. **Embracing change.** If a client thinks a product needs to be changed, programmers should support this decision and plan how to implement new requirements.
5. **Quality work.** A team that works well, makes a valuable product and feels proud of it.

Having discussed the main values and principles of XP, let's take a closer look at the practices inherent in this framework.

Extreme programming practices

The practices of XP are a set of specific rules and methods that distinguishes it from other methodologies. When used in conjunction, they reinforce each other, help mitigate the risks of the development process, and lead to the expected high-quality result. XP suggests using 12 practices while developing software which can be clustered into four groups.


EXTREME PROGRAMMING PRACTICES

Group	Practices
Feedback	✓ Test-Driven Development ✓ The Planning Game ✓ On-site Customer ✓ Pair Programming
Continual Process	✓ Continuous Integration ✓ Code Refactoring ✓ Small Releases
Code understanding	✓ Simple Design ✓ Collective Code Ownership ✓ System Metaphor ✓ Coding Standards
Work conditions	✓ 40-Hour Week

altexsoft
software r&d engineering

XP main practices

Test-Driven Development

Is it possible to write a clear code quickly? The answer is yes, according to XP practitioners. The quality of software derives from short development cycles that, in turn, allow for receiving frequent feedback. And valuable feedback comes from good testing. XP teams practice test-driven development technique (TDD) that entails writing an automated unit test before the code itself. According to this approach, every piece of code must pass the test to be released. So, software engineers thereby focus on writing code that can accomplish the needed function. That's the way TDD allows programmers to use immediate feedback to produce reliable software. You can learn more about improving software testing in our dedicated article.

The Planning Game

This is a meeting that occurs at the beginning of an iteration cycle. The development team and the customer get together to discuss and approve a product's features. At the end of the planning game, developers plan for the upcoming iteration and release, assigning tasks for each of them.

On-site Customer

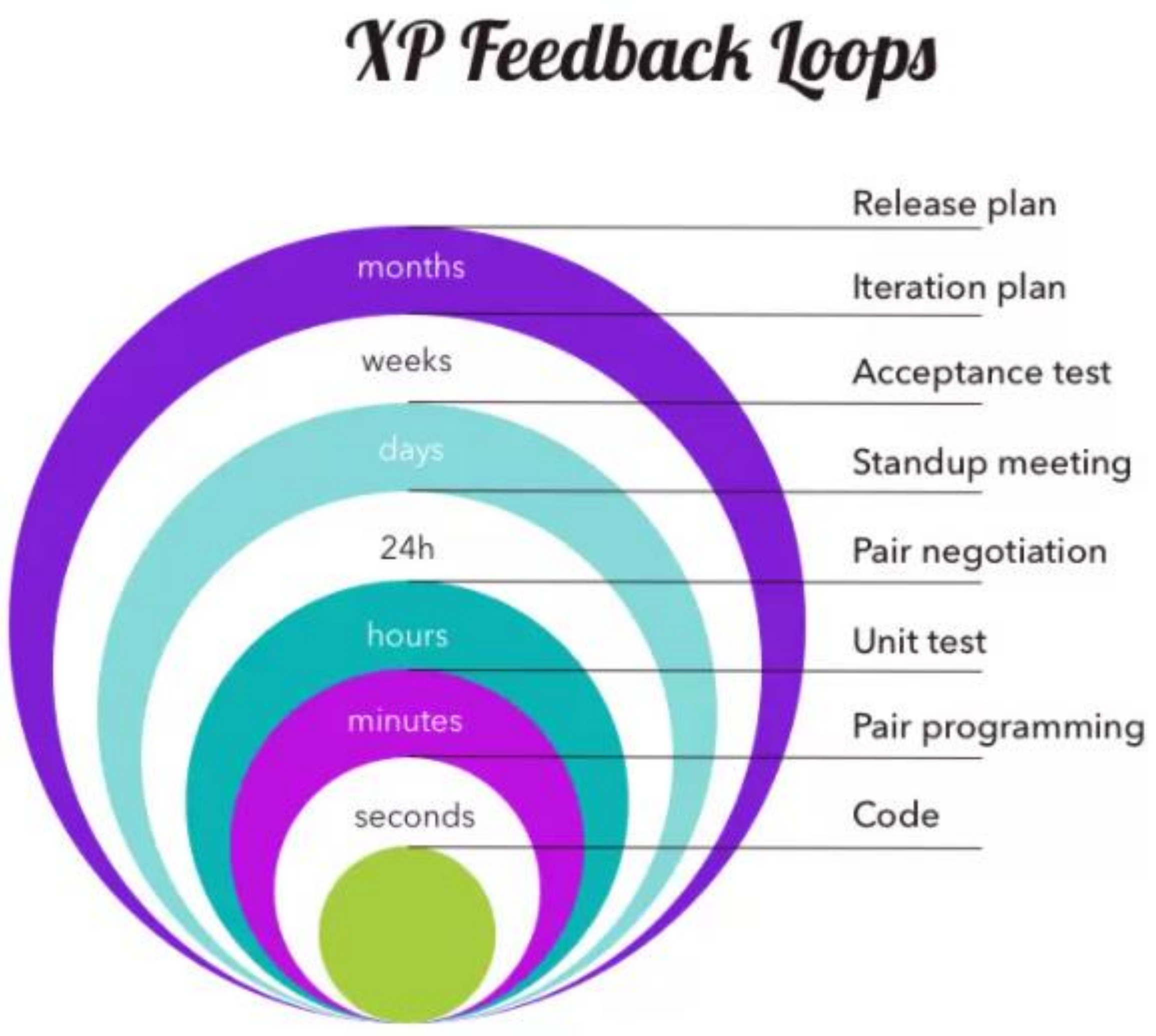
As we already mentioned, according to XP, the end customer should fully participate in development. The customer should be present all the time to answer team questions, set priorities, and resolve disputes if necessary.

Pair Programming

This practice requires two programmers to work jointly on the same code. While the first developer focuses on writing, the other one reviews code, suggests improvements, and fixes mistakes along the way. Such teamwork results in high-quality software and faster knowledge sharing but takes about 15 percent more time. In this regard, it's more reasonable trying pair programming for long-term projects.

Code Refactoring

To deliver business value with well-designed software in every short iteration, XP teams also use refactoring. The goal of this technique is to continuously improve code. Refactoring is about removing redundancy, eliminating unnecessary functions, increasing code coherency, and at the same time decoupling elements. *Keep your code clean and simple, so you can easily understand and modify it when required* would be the advice of any XP team member.



Pair programming in XP iteration cycle, source: extremeprogramming.org

Continuous Integration

Developers always keep the system fully integrated. XP teams take iterative development to another level because they commit code multiple times a day, which is also called continuous delivery. XP practitioners understand the importance of communication. Programmers discuss which parts of the code can be re-used or shared. This way, they know exactly what functionality they need to develop. The policy of shared code helps eliminate integration problems. In addition, automated testing allows developers to detect and fix errors before deployment.

Small Releases

This practice suggests releasing the MVP quickly and further developing the product by making small and incremental updates. Small releases allow developers to frequently receive feedback, detect bugs early, and monitor how the product works in production. One of the methods of doing so is the continuous integration practice (CI) we mentioned before.

Simple Design

The best design for software is the simplest one that works. If any complexity is found, it should be removed. The right design should pass all tests, have no duplicate code, and contain the fewest possible methods and classes. It should also clearly reflect the programmer's intent.

XP practitioners highlight that chances to simplify design are higher after the product has been in production for some time. Don Wells advises writing code for those features you plan to implement right away rather than writing it in advance for other future features: "The best approach is to create code only for the features you are implementing while you search for enough knowledge to reveal the simplest design. Then refactor incrementally to implement your new understanding and design."

Coding Standards

A team must have common sets of coding practices, using the same formats and styles for code writing. Application of standards allows all team members to read, share, and refactor code with ease, track who worked on certain pieces of code, as well as make the learning faster for other programmers. Code written according to the same rules encourages collective ownership.

Collective Code Ownership

This practice declares a whole team's responsibility for the design of a system. Each team member can review and update code. Developers that have access to code won't get into a situation in which they don't know the right place to add a new feature. The practice helps avoid code duplication. The implementation of collective code ownership encourages the team to cooperate more and feel free to bring new ideas.

System Metaphor

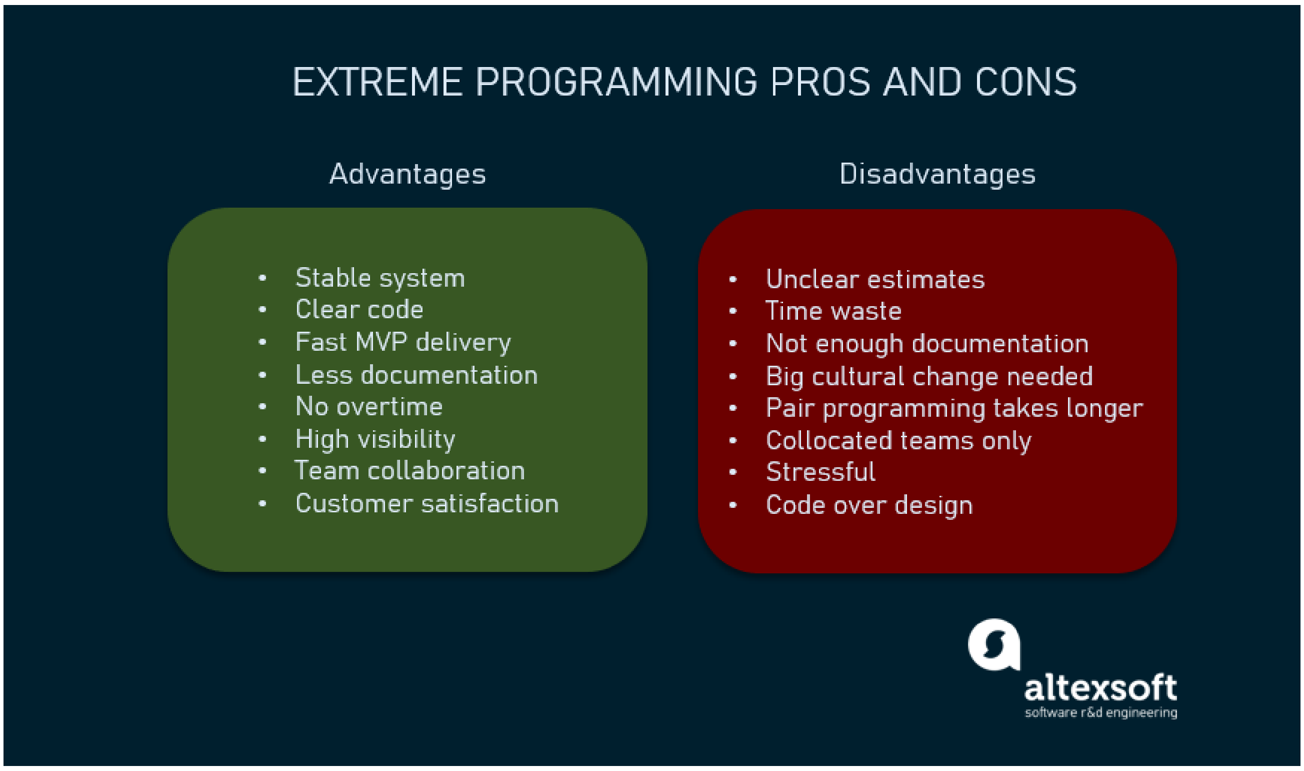
System metaphor stands for a simple design that has a set of certain qualities. First, a design and its structure must be understandable to new people. They should be able to start working on it without spending too much time examining specifications. Second, the naming of classes and methods should be coherent. Developers should aim at naming an object as if it already existed, which makes the overall system design understandable.

40-Hour Week

XP projects require developers to work fast, be efficient, and sustain the product's quality. To adhere to these requirements, they should feel well and rested. Keeping the work-life balance prevents professionals from burnout. In XP, the optimal number of work hours must not exceed 45 hours a week. One overtime a week is possible only if there will be none the week after.

Advantages and disadvantages of XP

XP practices have been debated upon for decades, as its approach and methods are rather controversial in a number of aspects and can't be applied in just any project. Here, we'll try to define the pros and cons of XP methodology.



EXTREME PROGRAMMING PROS AND CONS

Advantages	Disadvantages
Stable system	Unclear estimates
Clear code	Time waste
Fast MVP delivery	Not enough documentation
Less documentation	Big cultural change needed
No overtime	Pair programming takes longer
High visibility	Collocated teams only
Team collaboration	Stressful
Customer satisfaction	Code over design

XP pros and cons in a nutshell

Extreme programming advantages

So, the XP framework can be beneficial and help reduce development time and costs for the following reasons:

- Continuous testing and refactoring practices help create **stable well-performing systems** with minimal debugging;

- Simplicity value implies creating a **clear, concise code** that is easy to read and change in the future if needed;
- The minimalistic iterative approach to development ensures that the workable **results can be delivered very soon** and only necessary features are built;
- **Documentation is reduced** as bulky requirements documents are substituted by user stories;
- No or **very little overtime** is practiced;
- Constant communication provides a high level of **visibility and accountability** and allows all team members to keep up with the project progress;
- Pair programming has proven to result in **higher-quality products** with fewer bugs; most research participants also reported enjoying such collaboration more and feeling more confident about their job;
- **Customer engagement ensures their satisfaction** as their participation in the development and testing process can directly influence the result, getting them exactly what they wanted.

Extreme programming disadvantages

On the other hand, XP has a number of disadvantages that have to be considered when deciding on which framework to choose for your next project:

- In many instances, the customer has no clear picture of the end result, which makes it almost **unrealistic to accurately estimate scope, cost, and time**;
- **Regular meetings with customers often take a great deal of time** that could instead be spent on actual code writing;
- **Documentation can be scarce** and lack clear requirements and specifications, leading to project scope creep;
- The rapid transition from traditional methods of software development to extreme programming demands significant **cultural and structural changes**;
- **Pair programming takes more time** and doesn't always work right due to the human factor and character incompatibility;
- **XP works best with collocated teams** and customers present in person to conduct face-to-face meetings, limiting its application with distributed teams;

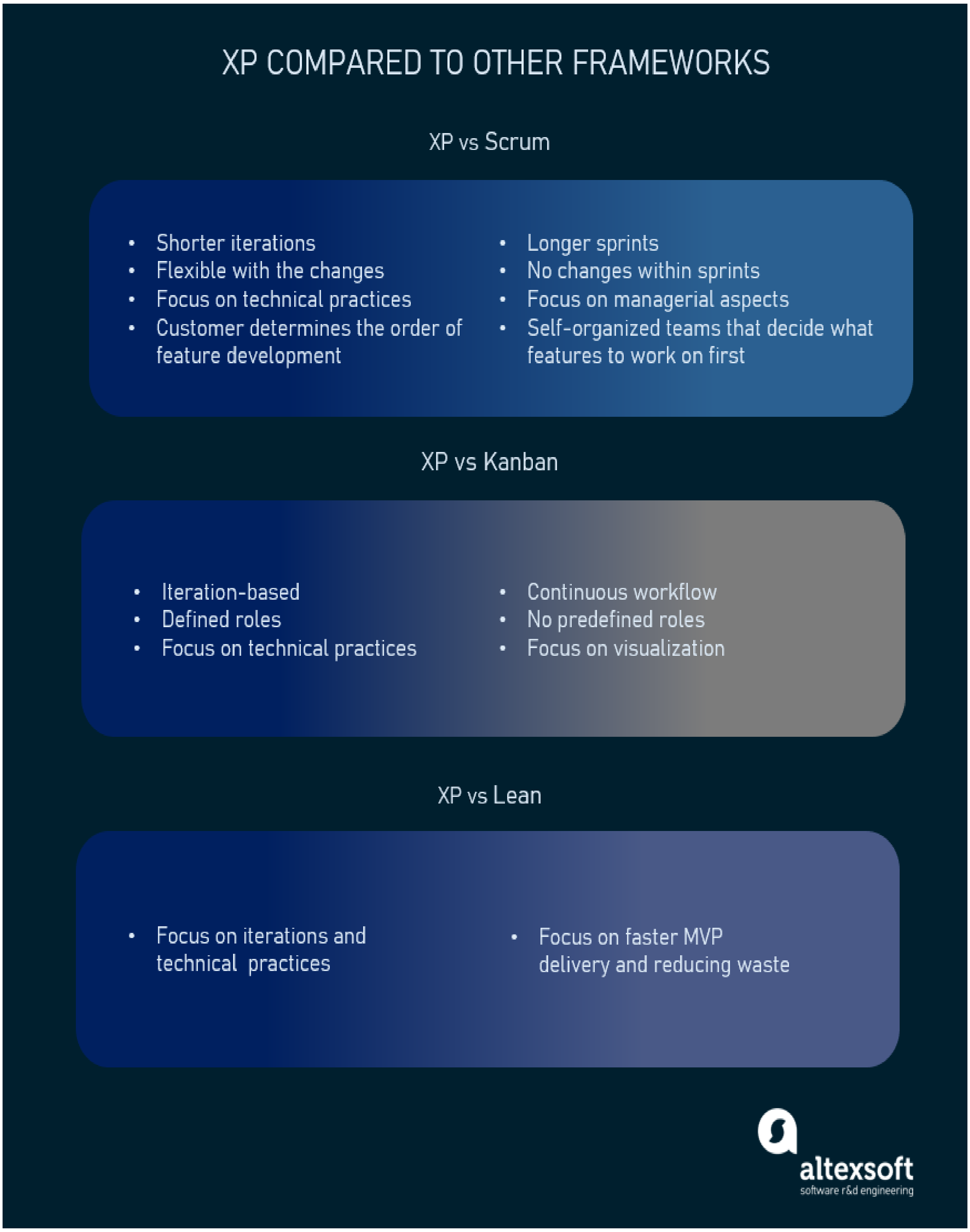
- Sometimes customers have neither the desire, time, nor expertise to participate in product development. Considering tight deadlines, it can become a **source of stress** as either no valuable feedback is provided, or a non-technical representative attempts to manage tech specialists with little or no knowledge on the process;
- Some authors also mention overfocusing on code over design, lack of quality assurance, code duplication, and poor results with inexperienced developers.

Any company can apply the XP principles in its projects; however, it's important to understand both the good and the bad sides. Read on to find out how XP is different from other methodologies and when applying its techniques would be the best choice.

Comparison of XP to other frameworks

As we mentioned above, XP is part of the agile methodology. It shares the main agile principles, i.e., frequent releases, short development cycles, constant communication with the customer, cross-functional teams, and so on. For this reason, XP is often confused with other popular Agile frameworks such as Scrum, Kanban, and Lean. Check our detailed whitepaper to get more in-depth information or the infographics for a quick summary of the main agile methods. Here, we'll briefly compare them and see what the main differences are.

But before we dive in, it's important to note that XP is not really a project management framework, even though a lot of its practices overlap with those from the project management domain. So, its primary focus is on the technical aspects of development and the implementation of specific practices rather than the management and organizational sides.



XP COMPARED TO OTHER FRAMEWORKS

XP vs Scrum

• Shorter iterations	• Longer sprints
• Flexible with the changes	• No changes within sprints
• Focus on technical practices	• Focus on managerial aspects
• Customer determines the order of feature development	• Self-organized teams that decide what features to work on first

XP vs Kanban

• Iteration-based	• Continuous workflow
• Defined roles	• No predefined roles
• Focus on technical practices	• Focus on visualization

XP vs Lean

• Focus on iterations and technical practices	• Focus on faster MVP delivery and reducing waste

altexsoft
software r&d engineering

XP vs Scrum, Kanban, and Lean in a nutshell

Extreme programming vs Scrum

Scrum is commonly associated with self-organizing teams. It also usually has sprints that are 2 to 4 weeks long, while XP iterations are shorter, taking 1 to 2 weeks. Besides, XP is much more flexible with possible changes within iterations, while Scrum doesn't allow any modifications after the sprint backlog is set. Another difference is that in XP the customer prioritizes features and decides on the order of their development, but in Scrum the team itself determines what to work on first.

Scrum's main roles are Product Owner, Scrum Master, and Scrum Team, which are different from those in XP.

However, there is no need to choose between XP and Scrum. Incorporating XP practices and Scrum techniques is considered quite effective with XP focusing on engineering aspects and Scrum organizing the process.

Extreme programming vs Kanban

Kanban puts a lot of focus on visualizing the development process and strictly limits the number of features developed at a time. It's also characterized by a continuous workflow while XP has separate iterations, even though both suggest small frequent releases and a high level of flexibility and adaptiveness to the changing requirements.

The roles in Kanban are not strictly defined.

Extreme programming vs Lean

It's hard to actually compare XP and Lean because the latter is more of a philosophy or approach to the development process and bringing value to the customer. Its core principles include eliminating waste, deciding as late as possible, delivering as early as possible, and so on. So, Lean's main focus is not on time-boxed iterations or specific engineering practices as in XP, but largely on a fast MVP delivery and reducing time waste.

When to use XP

Now that we discussed the XP methodology pros and cons and identified its place among other agile frameworks, we can talk about the cases when it's applicable. It's important to make sure a company's size, structure, and expertise, as well as the staff's knowledge base allow for applying XP practices. These are the factors to consider.

Highly-adaptive development. Some systems don't have constant functionality features and implies frequent changes. XP was designed to help development teams adapt to fast-changing requirements.

Risky projects. Teams applying XP practices are more likely to avoid problems connected with working on a new system, especially when a customer sets strict deadlines for a project. Additionally, a high level of customer engagement reduces the risk of their not accepting the end product.

Small teams. XP practices are efficient for teams that don’t exceed 12 people. Managing such groups is usually easier, communication is more efficient, and it takes less time to conduct meetings and brainstorming sessions.

Automated testing. Another factor that can influence the choice of XP is the developers’ ability to create and run unit tests, as well as availability of the necessary testing tools.

Readiness to accept new culture and knowledge. XP is different from traditional approaches to software development, and the way some of its practices should be implemented might not be obvious. So, it’s important that your organization and team members are ready to embrace change. It’s also worth inviting an experienced coach if you don’t have previous involvement with XP.

Customer participation. As XP requires customers, developers and managers to work side-by-side, make sure your client is always available to provide input until a project ends.

Agility principles are becoming increasingly popular as they prove their effectiveness. Even though extreme programming is not the most widespread methodology, it offers a lot of sensible practices that can benefit software development and are worth considering for implementation in your projects.