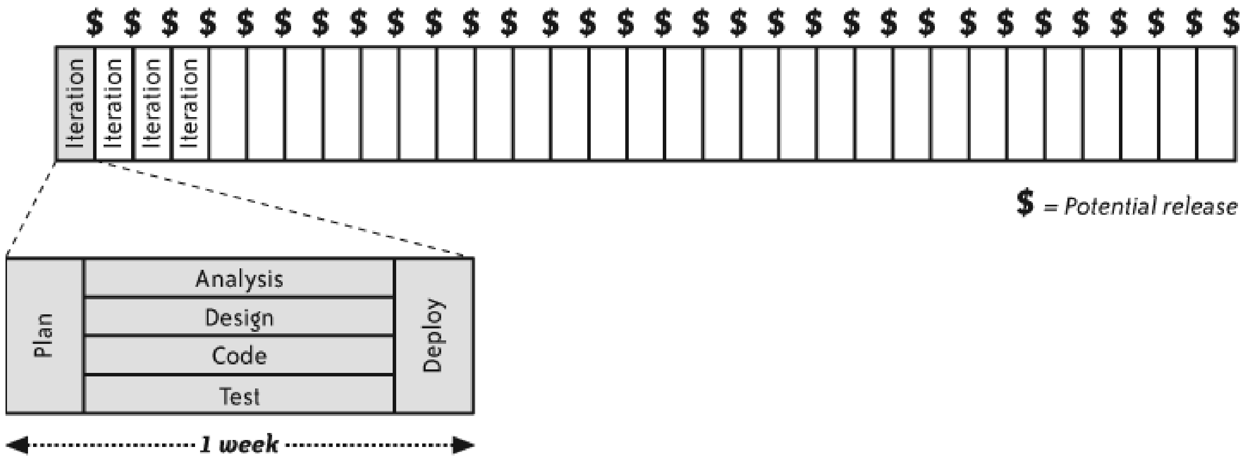
UNDERSTANDING XP

Extreme Programming (XP) is an Agile software development methodology that focuses on high-quality code, fast feedback, and close collaboration with customers.

XP Life cycle



XP emphasizes face-to-face collaboration. This is so effective in eliminating communication delays and misunderstandings that the team no longer needs distinct phases. This allows them to work on all activities every day–with simultaneous phases.

Planning:

Every XP team includes several business experts, the on-site customers, who are responsible for making business decisions. The on-site customers point the project in the right direction by clarifying the project vision, creating stories, constructing a release plan, and managing risks.

The planning activity (also called the planning game) begins with listening, a requirements gathering activity that enables the technical members of the XP team to understand the business context for the software and to get a broad feel for required output and major features and functionality.

Listening leads to the creation of stories by the customers and assigns a value(priority) to the story based on the overall business value of the feature or function. User stories describe required output, features, and functionality for software to be built. Members of the XP team then assess each story and assign a cost measured in development weeks to it. If the story is estimated to require more than three development weeks, the customer is asked to split the story into smaller stories and the assignment of value and cost occurs again.

Customers and developers work together to decide how to group stories into the next release (the next software increment) to be developed by the XP team. Once a basic commitment (agreement on stories to be included, delivery date, and other project matters) is made for a release, the XP team orders the stories that will be developed in one of three ways:

(1) all stories will be implemented immediately (within a few weeks),
(2) the stories with highest value will be moved up in the schedule and implemented first, or
(3) the riskiest stories will be moved up in the schedule and implemented first. Together, the team strives to create small, frequent releases that maximize value.

During the development of project, customers continue to review and improve the vision and the release plan to account for new opportunities and unexpected events. In addition to the overall release plan, the team creates a detailed plan for the upcoming week at the beginning of each iteration. The team touches base every day in a brief stand-up meeting, and its informative workspace keeps everyone informed about the project status.

Analysis:

On-site customers are responsible for figuring out the requirements for the software. To do so, they use their own knowledge as customers combined with traditional requirements gathering techniques. Customers are responsible for providing the required information to the programmers. They figure out the general requirements for a story before the programmers estimate it and the detailed requirements before the programmers implement it. If the requirements are tricky and difficult to understand, customers formalize them with the help of testers by creating customer tests. Customers and testers create the customer tests for a story around the same time that programmers implement the story.

Design:

Developers take the user stories and design the code architecture to meet the customer's expectations. This includes deciding on the programming language, environment, libraries, and frameworks to be used in software development. XP uses incremental design and architecture to continuously create and improve the design in small steps.

<u>Coding</u>:

Extreme Programming (XP) promotes pair programming i.e**. t**wo developers work together at one workstation, enhancing code quality and knowledge sharing. They write tests before coding to ensure functionality from the beginning, Test Driven Development (TDD), an activity that tightly links together testing, coding, design, and architecture.

Programmers are also responsible for managing their development environment. They use a version control system for configuration management and maintain their own automated build. Programmers integrate their code every few hours and ensure that every integration is technically capable of deployment.

<u>Testing</u>:

XP includes a sophisticated suite of testing practices. Well-functioning XP teams produce only a handful of bugs per month in completed work. With TDD, programmers produces automated unit and integration tests. Testers use exploratory testing to look for surprises and gaps in the software.

When the testers find a bug, the team conducts root-cause analysis and considers how to improve their process to prevent similar bugs from occuring in the future. Testers also explore the software's nonfunctional characteristics, such as performance and stability. Customers then use this information to decide whether to create additional stories.

Every time programmers integrate (once every few hours), they run the entire suite of regression tests to check if anything has broken.

<u>Deployment</u>:

XP teams keep their software ready to deploy at the end of any iteration. They deploy the software to internal stakeholders every week in preparation for the weekly iteration demo. Deployment to real customers is scheduled according to business needs.

As long as the team is active, it maintains the software it has released. When the project ends, the team may hand off maintenance duties to another team. In this case, the team creates documentation and provides training as necessary during its last few weeks.

XP Team

Different people know:

- How to design and program the software (programmers, designers, and architects)
- Why the software is important (product manager)
- The rules the software should follow (domain experts)
- How the software should behave (interaction designers)
- How the user interface should look (graphic designers)
- Where defects are likely to hide (testers)
- How to interact with the rest of the company (project manager)
- Where to improve work habits (coach)

All of this knowledge is necessary for project success.

XP creates cross-functionality teams composed of diverse people who can fulfill all the team’s roles.

The Whole Team:

XP teams sit together in an open workspace. At the beginning of each iteration, the team meets for a series of activities: an iteration demo, a retrospective, and iteration planning which takes 2-4 hours in total. The team also meets for daily stand-up meetings, which usually take five to ten minutes each. Sometimes, meeting is as informal as somebody standing up and announcing across the shared workspace that he would like to discuss an issue.

This self-organization is a hallmark of agile teams.

On-site customers:

On-site customers are responsible for defining software requirements and for determining what stakeholders find value. Customers most important activity is release planning. Customers need to promote the project’s vision: identify features and stories, determine how to group features into small , frequent releases, manage risks, create an achievable plan by coordinating with programmers.

On-site customers may or may not be real customers. Customers are responsible for refining their plans by soliciting feedback from real customers and other stakeholders through the weekly iteration demo.

In addition to planning, customers are responsible for providing programmers with requirements details upon request. Customers themselves act as living requirements documents, researching information in time for programmer use and providing it as needed. Customers also help communicate requirements by creating mock-ups, reviewing work in progress, and creating detailed customer tests that clarify complex business rules.

A great team will produce technically excellent software without on-site customers, but to truly succeed, your software must also bring value to its investors. This requires the perspective of on-site customers.

The Product Manager:

The job of product manager is to maintain and promote the product vision. The product manager does the following activities:

- documents the vision,
- shares it with stakeholders,
- incorporates feedback,
- generates features and stories,
- sets priorities for release planning,
- provides direction for the team's on-site customers,
- reviews work in progress,
- leads iteration demo,
- involves real customers, and
- deals with organizational politics

The best product managers have deep understandings of their markets. Good product managers have an intuitive understanding of what the software will provide and why it' s the most important thing their project teams can do with their time. A great product manager must have the authority to make difficult trade-off decisions about what goes into the product and what stays out.

A product manager should participate in every retrospective, every iteration demo, and most release planning sessions. In addition to maintaining and promoting the product vision, product managers are also often responsible for ensuring a successful deployment of the product to market. That may mean advertising and promotion, setting up training, and so forth.

Domain Experts:

Most software is built for a specific industry with its own business rules. These rules are domain rules, and knowledge of these rules is domain knowledge. Most programmers have gaps in their domain knowledge, even if they have worked in an industry for years.

The team's domain experts are responsible for figuring out these details and having the answers at their fingertips. Domain experts, also known as subject matter experts, are experts in their field. Examples include financial analysts and PhD chemists.

Domain experts spend most of their time with the team, figuring out the details of upcoming stories and standing ready to answer questions when programmers ask. For complex rules, they create customer tests.

Interaction Designers:

The user interface is the public face of the product. Users judge the product's quality based on the UI. Interaction Designers help define the product UI. They focus on understanding users, their needs and how they will interact with the product. They perform tasks as interviewing users, reviewing paper prototypes with users.

Interaction designers divide their time between with the team and working with users. During each iteration, they help the team create mock-ups of UI elements for that iteration's stories. As each story approaches completion, they review the look and feel of the UI and confirm that it works as expected.

Business Analysts:

Business analysts augment a team that already contains a product manager and domain experts. The analyst continues to clarify and refine customer needs, but the analyst does so in support of the other on-site customers, not as a replacement for them. Analysts help customers remember important things and explain technical choices in business-friendly words.

Programmers:

Most of an XP team is made up of software developers with different specialties, and all of them write working code. To emphasize this, XP refers to all developers as programmers.

Ideally, an XP team has 4–10 programmers with a mix of skills, including at least one experienced senior developer or architect who is comfortable coding. This helps the team succeed with XP's incremental design and architecture.

Customers focus on maximizing product value, while programmers focus on minimizing cost. Programmers find the most effective way to deliver the planned stories by estimating effort, suggesting alternatives, and helping create a realistic plan through the planning game.

Programmers spend most of their time pair programming. Using test-driven development, they write tests, implement code, refactor, and incrementally design and architect the application. They pay careful attention to design quality, and they are keenly aware of technical debt and its impact on development time and future maintenance costs.

They maintain a ten-minute build that can build a complete release package at any time. They use version control and practice continuous integration, keeping all but the last few hours' work integrated and passing its tests.

At the beginning of the project, the programmers establish coding standards that allow them to collectively share responsibility for the code. Programmers have the right and the responsibility to fix any problem they see, no matter which part of the application it touches.

Finally, programmers help ensure the long-term maintainability of the product by providing documentation at appropriate times.

Designer and Architects:

Everybody codes on an XP team, and everybody designs. Expert designers and architects contribute by guiding the team's incremental design and architecture efforts and by helping team members see ways of simplifying complex designs. They act as peers that is, as programmers rather than teachers, guiding rather than dictating.

Technical Specialists:

The programmers could include a database designer, a security expert, or a network architect. XP programmers are generalizing specialists. Although each person has his own area of expertise, everybody is expected to work on any part of the system that needs attention.

Testers:

Testers help XP teams produce quality results from the beginning. Testers apply their critical thinking skills to help customers consider all possibilities when envisioning the product. They help customers identify holes in the requirements and assist in customer testing.

Testers also provide information about the software's nonfunctional characteristics, such as performance, scalability, and stability, by using both exploratory testing and long-running automated tests.

However, testers don't exhaustively test the software for bugs. Rather than relying on testers to find bugs for programmers to fix, the team should produce nearly bug-free code on their own. When testers find bugs, they help the rest of the team figure out what went wrong so that the team as a whole can prevent those kinds of bugs from occurring in the future.

Some XP teams don’t include dedicated testers. If you don’t have testers on your team, programmers and customers should share this role.

Coaches:

XP teams are self-organizing, meaning team members decide how best to contribute at any moment. Instead of assigning tasks like traditional managers, XP leaders called Coaches support the team and help it perform at its best.

Coaches help start the process by setting up a shared workspace and ensuring the right people are on the team. They create conditions for productive work and help build an informative workspace. They also act as a bridge between the team and the rest of the organization by building trust, handling communication, and managing required reporting.

In addition, Coaches help team members stay disciplined and effectively follow challenging practices such as risk management, test-driven development, and incremental design and architecture.

The Programmer-Coach:

Every team needs a programmer-coach to help the other programmers with XP’s technical practices. Programmer-coaches are often senior developers and may have titles such as “technical lead” or “architect.” They can even be functional managers. While some programmer-coaches make good all-around coaches, others require the assistance of a project manager. Programmer-coaches also act as normal programmers and participate fully in software development.

The project manager:

Project managers help the team work with the rest of the organization. They are usually good at coaching non-programming practices. Some functional managers fit into this role as well. However, most project managers lack the technical expertise to coach XP' s programming practices, which necessitates the assistance of a programmer-coach.

Other team members:

- The Project Community
- Stakeholders
- The executive sponsor

<u>XP Concepts</u>

<u>Refactoring</u>: Refactoring is the process of changing the structure of code , rephrasing it without changing its meaning or behavior. It is used to improve code quality.

<u>Technical Debt</u>: Technical debt is the total amount of less-than-perfect design and implementation decisions in the project. This includes quick and dirty hacks intended just to get something working right now. Technical debt can even come from development practices such as an unwieldy build process or incomplete test coverage.

The bill for this debt often comes in the form of higher maintenance costs. There may not be a single lump sum to pay, but simple tasks that ought to take minutes may stretch into hours or afternoons. Unchecked technical debt makes the software more expensive to modify than to re-implement.

The key to managing it is to be constantly vigilant. Avoid shortcuts, use simple design, refactor relentlessly... in short, apply XP's development practices.

<u>TimeBoxing</u>: Timeboxing is a powerful time management technique where you allocate a fixed unit of time for a specific activity. In XP, teams work in short**,** fixed-length iterations (usually 1–2 weeks). Each iteration is a timebox. Timeboxing is used in iterations, planning, retrospectives, meetings. Timeboxing helps to increase focus, for better time management, enhances prioritization and improves efficiency.

Last Responsible Moment: Last Responsible Moment is the moment at which failing to make a decision eliminates an important alternative. XP uses Last Responsible Moment to reduce wrong decisions made with incomplete information, to allow better technical and business choices. By delaying decisions until this crucial point, you increase the accuracy of your decisions, decrease your workload, and decrease the impact of changes.

User Stories: A user story is a very high-level definition of a requirement, containing just enough information so that the developers can produce a reasonable estimate of the effort to implement it.

Iterations: An iteration is the full cycle of design-code-verify-release practiced by XP teams. It's a timebox that is usually one to three weeks long. Each iteration begins with the customer selecting which stories the team will implement during the iteration, and it ends with the team producing software that the customer can install and use. Though it may seem that small and frequent iterations contain a lot of planning overhead, the amount of planning tends to be proportional to the length of the iteration.

Velocity: Velocity is a simple way of mapping estimates to the calendar. It is the total of the estimates for the stories i.e. story points completed in an iteration. In general, the team should be able to achieve the same velocity in every iteration. Velocity is used in XP to decide how many stories to take into the next iteration, to predict how many iterations are needed to complete a release.

Theory of Constraints: Theory of Constraints (ToC) in Extreme Programming (XP) is about finding the biggest bottleneck in the development process and improving it to increase overall throughput. Many software teams can only complete their projects as quickly as the programmers can program them regardless of how much work testers and customers do. In such projects, programmers are the constraint. Legacy projects sometimes have a constraint of testing, not programming. The responsibility for estimates and velocity always goes to the constraint.

Mindfulness: The ability to respond effectively to change. Requires that everyone pay attention to the process and practices of development. This is mindfulness.

Adopting XP

Before adopting XP, one has to decide whether it is appropriate for the situation. XP's applicability is based on organizations and people, not types of projects.

Knowing whether XP is suitable

The following prerequisites and recommendations have to be met to practice XP.

Prerequisite1: Management Support
To practice XP, the following are needed:

- A common workspace with pairing stations
- Team members solely allocated to the XP project
- A product manager, an on-site customer and Integrated Testers

If the management is not supportive to adopt XP, think about how management cares about organizational success and personal success and how XP helps to achieve the success. Talk in terms of your managers' ideas of success. Even if management refuses, then XP is not appropriate for the team.

Prerequisite2: Team Agreement
If team members don't want to use XP, it's not likely to work. It's never a good idea to force someone to practice XP against his will. If only one or two people refuse to use XP, and they're interested in working on another project, let them transfer so the rest of the team can use XP. If no such project is available, or if a significant portion of the team is against using XP, don't use it. One way to help people agree to try XP is to promise to revisit the decision on a specific date. At that point, if the team doesn't want to continue using XP, stop.

Prerequisite3: Colocated team
XP relies on fast, high-bandwidth communication for many of its practices. In order to achieve that communication, your team members needs to sit together in the same room.
If the team is not colocated then talk with mentor.

Prerequisite4: On-site Customers
On-site customers are critical to the success of an XP team. They, led by the product manager, determine which features the team will develop. Of all the on-

site customers, the product manager is likely the most important who makes the final determination of value.

If there is an experienced product manager who makes high-level decisions about features and priorities, but who isn't available to sit with the team full-time, then a business analyst or one of the other on-site customers can be asked to act as a proxy. The proxy's job is to act in the product manager's stead to make decisions about details while following the actual product manager's high-level decisions.

If product manager is not available, someone from the development team can play the part, but it is a dangerous approach because this person is unlikely to have the business expertise necessary to deliver an organizational success. Better talk with your mentor about how to compensate.

If no on-site customer available, product manager, domain expert or interaction designer can play the role of on-site customer.

As long as somebody is playing the on-site customer role, you can use XP. However, the less expertise on-site customers, the more risk there is to the value of software.

Prerequisite5: The Right Team Size

Teams with fewer than four programmers are less likely to have the intellectual diversity they need. They will also have trouble using pair programming, an important support mechanism in XP. Large teams face coordination challenges. Although experienced teams can handle those challenges smoothly, a new XP team will struggle.

If there are no even pairs, add or drop one programmer so you have even pairs.

If the team has more than 7 programmers, then hire an experienced XP coach to lead the team through the transition.

If there are many solo developers, combine 4-7 of these programmers into a single XP team that works on one project at a time, which allows it to complete projects more quickly. By working together, senior developers have the opportunity to mentor junior developers.

Prerequisite6: Use All the Practices

XP doesn't require perfection. It's OK if you accidentally misapply a practice from time to time but it rarely works well if you arbitrarily remove pieces.

If you are sure a practice won't work, you need to replace it. Replacing practices requires continuous refinement and an in-depth understanding of XP. Ask your mentor for help and consider hiring an experienced XP coach.

Recommendation1: A Brand-New Codebase
XP teams put a lot of effort into keeping their code clean and easy to change. If you have a brand-new codebase, this is easy to do. If you have to work with existing code, you can still practice XP, but it will be more difficult. Even well-maintained code is unlikely to have the simple design and suite of automated unit tests that XP requires.

Recommendation2: String Design Skills
At least one person on the team, preferably a natural leader needs to have strong design skills.

Recommendation3: A Language that is easy to Refactor
XP relies on refactoring to continuously improve existing designs, so any language that makes refactoring difficult will make XP difficult. Of the currently popular languages, object-oriented and dynamic languages with garbage collection are the easiest to refactor. C and C++, for example, are more difficult to refactor.

Recommendation4: An Experienced Programmer-Coach
XP relies on self-organizing teams.
The best coaches are natural leaders, people who remind others to do the right thing by virtue of who they are rather than the orders they give. Coach also needs to be an experienced programmer so he/she can help the team with XP’s technical practices. If no coach in the team, then ask programmers to choose a coach.

Recommendation5: A Friendly and Cohesive Team
XP requires that everybody work together to meet team goals.

Implementing XP

Applying XP to a Brand-New Project

In a new XP project, the first few weeks are usually chaotic as the team learns to work together, plans releases, and sets up technical tools. Some teams try to avoid this by doing extra planning and setup before starting, but XP encourages beginning real iterations right away. Planning and building infrastructure should happen gradually throughout the project, not all at once at the start.

When starting an XP project, the first task is to plan the first iteration. Since there is no release plan yet, the team chooses one feature that will definitely be in the first release. They then create a few essential stories for that feature that form a "vertical slice" of the system showing a basic but complete flow.

The first stories should be very simple. Because programmers are still setting up the technical infrastructure, their estimates will be high and the stories will deliver minimal functionality. For example, a report may only show headers, an installer may just be a zip file, and a screen may only display a logo.

The simple initial stories help the team discover more detailed stories. In the first planning session, brainstorm about 10–20 stories and have programmers estimate them. Choose stories which programmers already understand so customers can focus on creating the release plan instead of answering questions.

Planning the first iteration is harder because the team doesn't know its velocity yet, so make a best guess. During the iteration, work on only one or two stories at a time and review progress daily to stay on track.

After planning, programmers set up the technical infrastructure, such as version control and integration systems, then begin working on stories. In the first iteration, it's best for programmers to work together as a group to reduce confusion and agree on basic conventions. After a few days, the team can split into pairs and work normally.

While programmers code, customers and testers should define the product vision and release plan. Over time, estimates become more accurate, velocity stabilizes, planning gets easier, and the initial chaos fades as the team settles into a steady rhythm.

Applying XP to an Existing Project

New (greenfield) projects can adopt all XP practices from the start and usually become stable within four to nine months. Legacy projects with existing code and no tests can also succeed with XP, but they need to adopt the practices gradually and will take longer to see results.

The Big Decision

In existing (legacy) projects, the biggest challenge when adopting XP is making time to reduce technical debt. Many teams takes shortcuts to meet deadlines, which creates hidden technical debt. To improve productivity and reduce bug production, teams must stop adding new debt and also spend extra time(slack) to fix existing debt. This initially lowers velocity. However, as technical debt is reduced, productivity improves, and over time the team's velocity will recover and eventually become higher than before, though this may take several months.

Setting aside time (slack) to reduce technical debt is difficult, but necessary. If teams keep adding technical debt, productivity will keep falling and defects will increase, making even small changes too costly. This can force organizations to abandon or completely rewrite the product. By investing in reducing technical debt early, product managers can turn a struggling legacy system into a sustainable, long-term asset.

Bring Structure

When improving a legacy project, the first step is to bring structure to the work. Many older projects suffer from poor or inconsistent planning. Start by introducing XP's structural practices, such as working in a shared space, using pair programming, and holding regular iteration planning and retrospectives.

Begin with short iterations, ideally two weeks, and adjust only if necessary with guidance from a mentor. Convert the existing project plan into story cards, even if the stories are not customer-focused at first. As the team becomes comfortable with iterations, the stories can be refined to better reflect customer value. Once these structural practices are in place, the team can gradually introduce XP's technical practices.

Reduce Technical Debt

The biggest problem facing legacy projects is usually excessive technical debt. To fix this, first stop creating new debt by setting up a quick build, using continuous integration, and adopting test-driven development. At the same time, reduce existing debt by adding extra slack in your iterations to clean up the code. Initially, progress may feel slow, but over time code quality and productivity will improve. Once quality rises, you can introduce the remaining XP practices and start organizing the bug backlog.

Organize backlog

Bug databases often contain to-dos, questions, feature requests, and real defects. Customers and testers should clean it up by removing duplicates and unimportant items, turning feature requests into stories, and keeping only genuine defects. This cleanup may take several iterations. For ongoing use, review and organize new entries daily to prevent clutter and keep the database useful.

Fix important bugs

Once your bug database is clean, decide for each bug whether to fix it now or defer it. Involve the product manager and have programmers estimate the effort for tricky bugs. Close or postpone the ones you won't fix this release. Convert the remaining bugs into stories, estimate them, and add them to the release plan. During the release, fix these bugs, prevent their causes, continue reducing technical debt, and start doing some root-cause analysis to avoid future problems.

Move testers forward

At first, testers spend most of their time on manual regression testing before each release. As programmers adopt test-driven development, automated regression tests will gradually reduce this workload. Use iteration slack to automate remaining manual tests, starting with end-to-end tests and later refactoring them into unit and integration tests. With fewer new bugs, testers can shift from after-the-fact testing to helping improve code quality from the start, working with customers to find requirement gaps and performing exploratory testing.

Emerge from the darkness

Following this process reduces technical debt, improves code quality, and eliminates defects, which gradually increases productivity. Progress may be slow at first and could take many months to reach near-zero new bugs, complete the regression suite, and fully integrate testers. As long as each iteration reduces debt, you'll steadily improve. Over time, you'll see more reliable estimates and a more enjoyable programming experience.