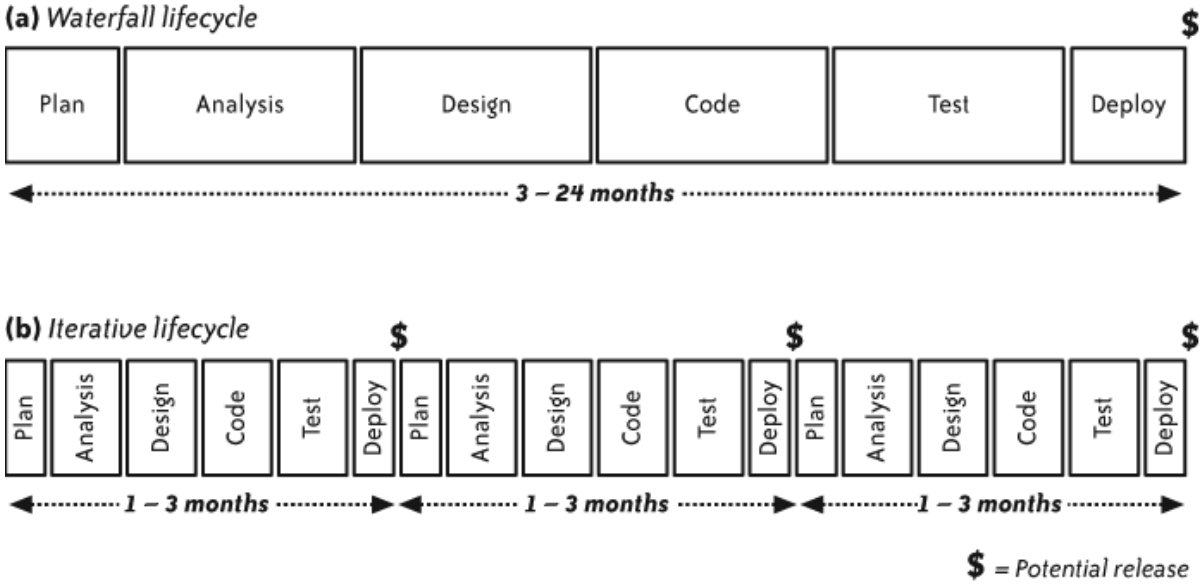
Agile Technologies

Module-2

Understanding XP: The XP Lifecycle, The XP Team, XP Concepts, Adopting XP: Is XP Right for Us?, Go!, Assess Your Agility

Overview of Extreme Programming, The Practices of Extreme Programming, Conclusion, Bibliography, Planning Initial Exploration, Release Planning, Iteration Planning, Defining "Done", Task Planning Iterating, Tracking.

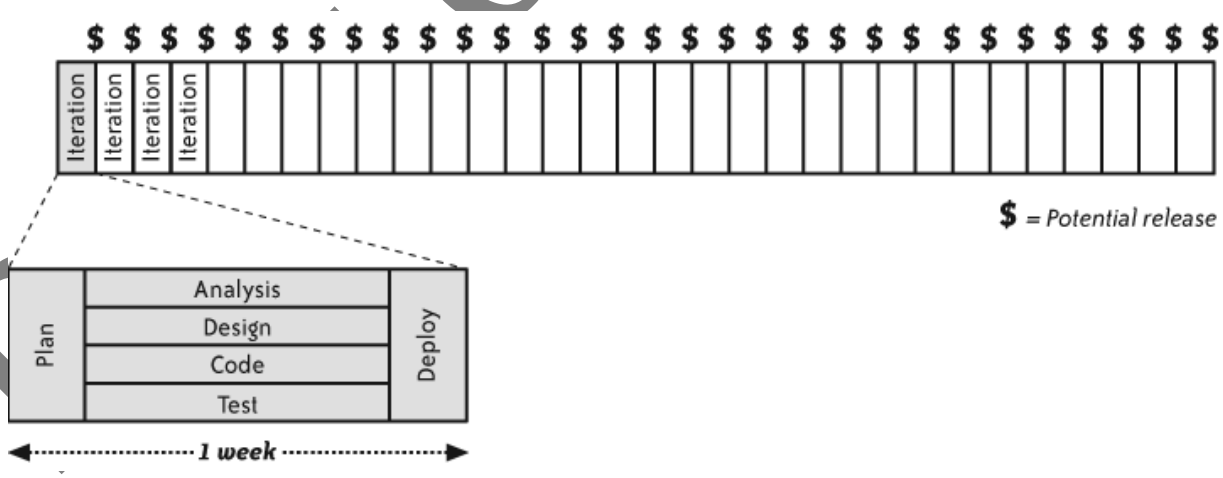
Understanding XP



The XP Lifecycle

Software projects do need more requirements, design, and testing—which is why XP teams work on these activities every day. Yes, every day.

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 —as shown below



How It Works

- XP teams perform nearly every software development activity simultaneously.

- XP does it by working in iterations: week-long increments of work. Every week, the team does a bit of release planning, a bit of design, a bit of coding, a bit of testing, and so forth.

- They work on stories: very small features, or parts of features, that have customer value. Every week, the team commits to delivering four to ten stories.

- Throughout the week, they work on all phases of development for each story. At the end of the week, they deploy their software for internal review.

The following sections show how traditional phase-based activities correspond to an XP iteration.

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. Programmers provide estimates and suggestions based on customer priorities

Together, the team strives to create small, frequent releases that maximize value.
In addition to the overall release plan, the team creates a detailed plan for the upcoming week at the beginning of each iteration, its informative workspace keeps everyone informed about the project status.

Analysis

On-site customers sit with the team full-time. 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. When programmers need information, they simply ask. Customers are responsible for organizing their work so they are ready when programmers ask for information.

Some requirements are tricky or difficult to understand. Customers formalize these requirements, with the assistance of testers, To assist in communication, programmers use a ubiquitous language in their design and code.

Design and coding

XP uses incremental design and architecture to continuously create and improve the design in small steps. This work is driven by test-driven development (TDD), To support this process, programmers work in pairs, and ensures that one person in each pair always has time to think about larger design issues.

Programmers are also responsible for managing their development environment. They use a version control system for configuration management. Programmers also maintain coding standards and share ownership of the code.

Testing

In XP each member of the team—programmers, customers, and testers—makes his own contribution to software quality.

Programmers provide the first line of defence with test-driven development. TDD produces automated unit and integration tests. These tests help ensure that the software does what the programmers intended. Likewise, customers tests help ensure that the programmers' intent matches customers' expectations. Customers review work in progress to ensure that the UI works the way they expect. Finally, testers help the team understand whether their efforts are in fact producing high-quality code. When the testers find a bug, the team conducts root-cause analysis and considers how to improve their process to prevent similar bugs from occurring in the future. Testers also explore the software's non-functional characteristics, such as performance and stability. Customers then use this information to decide whether to create additional stories. When bugs are found, programmers create automated tests to show that the bugs have been resolved.

Deployment

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.

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.

The XP Team

Team software development requires the information to spread out among many members of the 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)

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 look back, and iteration planning. These typically take two to four hours in total. The team also meets for **daily stand-up meetings,** which usually take five to ten minutes each.

On-Site Customers

- On-site customers—often just called customers—are responsible for defining the software the team builds.

- Customers' most important activity is release planning. Customers need to highlight the project's vision; identify features and stories; determine how to group features into small, frequent releases; manage risks; and create an achievable plan by coordinating with programmers.

- On-site customers may or may not be real customers, regardless; customers are responsible for refining their plans by providing feedback from real customers and other stakeholders. One of the venues for this feedback is the weekly iteration demo.

- In addition to planning, customers are responsible for providing programmers with
- requirements details upon request.

- Typically, product managers, domain experts, interaction designers, and business analysts play the role of the on-site customer.

The product manager (aka product owner)

- The product manager has only one job on an XP project, that job is to maintain and promote the product vision.

- In practice, this means documenting the vision, sharing it with stakeholders, incorporating feedback, generating features and stories, setting priorities for

release planning, providing direction for the team's on-site customers, reviewing work in progress, leading iteration demos, involving real customers, and dealing with organizational politics.

- The best product managers have deep understandings of their markets.

Domain experts (aka subject matter experts)

- Most software operates in a particular industry, such as finance, that has its own specialized rules for doing business. To succeed in that industry, the software must implement those rules faithfully and exactly. These rules are domain rules, and knowledge of these rules is domain knowledge.
- Most programmers have gaps in their domain knowledge.
- 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.
- 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.

Interaction designers

- The user interface is the public face of the product. For many users, the UI is the product. They judge the product's quality solely on their perception of the UI.
- Interaction designers help define the product UI. Their job focuses on understanding users, their needs, and how they will interact with the product. They perform such tasks as interviewing users, creating user personas, reviewing paper prototypes with users, and observing usage of actual software.

Business analysts

On an XP team, 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 think of details they might otherwise forget and help programmers express technical trade-offs in business terms.

Programmers

- The bulk of the XP team consists of software developers in a variety of specialties. Each of these developers contributes directly to creating working code.
- If the customers' job is to maximize the value of the product, then the programmers' job is to minimize its cost.
- Programmers spend most of their time pair programming. Using test-driven development, they write tests, implement code and incrementally design and architect the application.
- With the help of the whole team, the programmers strive to produce no bugs in completed software.
- 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.
- Programmers rely on customers for information about the software to be built.

Designers and architects

Expert designers and architects are necessary. They contribute by guiding the team's incremental design and architecture efforts and by helping team members see ways of simplifying complex designs.

Technical specialists

- In XP the "programmer" role includes other software development roles. 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.
- They help customers identify holes in the requirements and assist in customer testing.*
- Testers also act as technical investigators for the team. They help the team identify whether it is successfully preventing bugs from reaching finished code.
- Testers also provide information about the software's non-functional characteristics, such as performance, scalability, and stability.
- 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.

Adopting XP

Before adopting XP, you need to decide whether it's appropriate for your situation. Often, people's default reaction to hearing about XP is to say, "**Well, of course that works for other teams, but it couldn't possibly work for us.**"

XP's applicability is based on organizations and people, not types of projects.

Is XP Right for Us?

You can adopt XP in many different conditions; here are some **prerequisites and recommendations** about your team's environment.

Prerequisite #1: Management Support

It's very difficult to use XP in the face of opposition from management. Active support is best., you will need the following: A common workspace with pairing ,Team members solely allocated to the XP project, A product manager, on-site customers, and integrated testers

If management isn't supportive.

If you want management to support your adoption of XP, they need to believe in its benefits. Think about what the decision-makers care about. What does an organizational success mean to your management? What does a *personal* success mean? How will adopting XP help them achieve those successes? What are the risks of trying XP, how will you mitigate those risks, and what makes XP worth the risks? Talk in terms of your managers' ideas of success, not your own success.

Prerequisite #2: Team Agreement

Just as important as management support is the team's agreement to use XP. If team members don't want to use XP, it's not likely to work. XP assumes good faith on the part of team members—there's no way to force the process on somebody who's resisting it.

If people resist...It's never a good idea to force someone to practice XP against his will. In the best case, he'll find some way to leave the team, quitting if necessary. In the worst case, he'll remain on the team and silently sabotage your efforts.

One way to help people agree to try XP is to promise to revisit the decision on a specific date. (Allow two or three months if you can.) At that point, if the team doesn't want to continue using XP, stop.

Prerequisite #3: A 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 your team isn't colocated...

Colocation makes a big difference in team effectiveness. Don't assume that your team can't sit together; be sure that bringing the team together is your first option.

Prerequisite #4: On-Site Customers

The on-site customers' decisions determine the value of the software.

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. In other words, their decisions determine the value of the software. Of all the on-site customers, the product manager is likely the most important. She makes the final determination of value. A good product manager will choose features that provide value to your organization.

If your product manager is too busy to be on-site...

If you have 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, you may be able to ask a business analyst or one of the other on-site customers to act as a *proxy*.

Prerequisite #5: The Right Team Size

For teams new to XP, however, I recommend 4 to 6 programmers and no more than 12 people on the team. I also recommend having an even number of programmers so that everyone can pair program.

If you don't have even pairs...

The easiest solution to this problem is to add or drop one programmer so you have even pairs.

Prerequisite #6: Use All the Practices

You may be tempted to ignore or remove some XP practices, particularly ones that make team members uncomfortable. Be careful of this. XP is designed to have very little waste. Nearly every practice directly contributes to the production of valuable software.

If practices don't fit...

You may think that some XP practices aren't appropriate for your organization. That may be true, but it's possible you just feel uncomfortable or unfamiliar with a practice.

Recommendation #1: A Brand-New Codebase

Easily changed code is vital to XP. If your code is cumbersome to change, you'll have difficulty with XP's technical practices, XP teams put a lot of effort into keeping their code clean and easy to change.

Recommendation #2: Strong Design Skills

Simple, easily changed design is XP's core enabler. This means at least one person on the team—preferably a natural leader—needs to have strong design skills.

Recommendation #3: A Language That's 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.

Recommendation #4: An Experienced Programmer-Coach

Your team needs a coach. 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. Your coach also needs to be an experienced programmer so she can help the team with XP's technical practices.

Recommendation #5: A Friendly and Cohesive Team

XP requires that everybody work together to meet team goals. There's no provision for someone to work in isolation, so it's best if team members enjoy working together.

Go!

- Are you ready to adopt XP?
- Great! Your first step is to arrange for your open workspace
- Find an appropriate project for the team to work on. Look for a project that's valuable
- At the same time, figure out who will be on your team. Talk with your project's executive sponsor and other stakeholders about whom to include as your on-site customers, Be sure your team members want to try XP. As you're forming your team, consider hiring an experienced XP coach to work with the team full-time

As your project start date draws near, you'll need supplies for the team's open workspace. The following is a good shopping list.

Equipment:

Pairing stations

- Noise-dampening partitions to define your team's workspace and prevent noise pollution.
- Plenty of wall-mounted whiteboards for discussions and charts . Ferrous (magnetic) whiteboards are best because you can stick index cards to them with magnets.
- Two big magnetic whiteboards for your release and iteration plans.
- A large plastic perpetual calendar (three months or more) for marking important dates and planned absences

- Any other equipment you normally use.

Software:

- A unit-testing tool such as the xUnit family
- An automated build tool such as the Ant family.
- Any other software you normally use.

Supplies:

- Pencils for index cards.
- Food.
- Dry-erase markers for whiteboards, water-based flip-chart markers for flip charts.
- Magnets for sticking papers to whiteboards.
- Any other supplies you normally use.

The Challenge of Change

It's a fact of life: change makes people uncomfortable. XP is probably a big change for your team. If you previously used a rigid, document-centric process, XP will seem loose and informal.

Expect team members and stakeholders to be uncomfortable. This discomfort can extend into the larger organization.

Discomfort and a feeling of chaos is normal for any team undergoing change, but that doesn't make it less challenging. Expect the chaotic feeling to continue for at least two months. Give yourselves four to nine months to feel truly comfortable with your new process. If you're adopting XP incrementally, it will take longer.

To survive the transformation, you need to know why you are making this change. What benefits does it provide to the organization? To the team? Most importantly, what benefits does it provide to each individual? As you struggle with the chaos of change, remember the benefits.

Our pledge to users, management, and other stakeholders.

We promise to:

- Make steady progress
- Finish the features that you consider most valuable first
- Show you working software that reflects our progress every week, on (day of week) at (time) in (location)

- Be honest and open with you about our successes, challenges, and what we can reasonably provide

Final Preparation

Before starting XP, it's a good idea to discuss working agreements—that is, which practices your team will follow .

Discuss your roles and what you expect from each other. It's best to hold these conversations as collaborative team discussions. Try to avoid assigning roles or giving people orders.

In the final weeks before starting your new XP project, review the practices

When you've finished these preparations, if you have your team is creating a new codebase from scratch—you're ready to go.

Well... yes. You can follow the incremental approach that legacy projects use,

Teams that adopt XP incrementally make substantial improvements, but it's the teams that adopt it all at once that really excel.

Be bold. You have the right people, the right workplace, and the will to succeed. Do it!

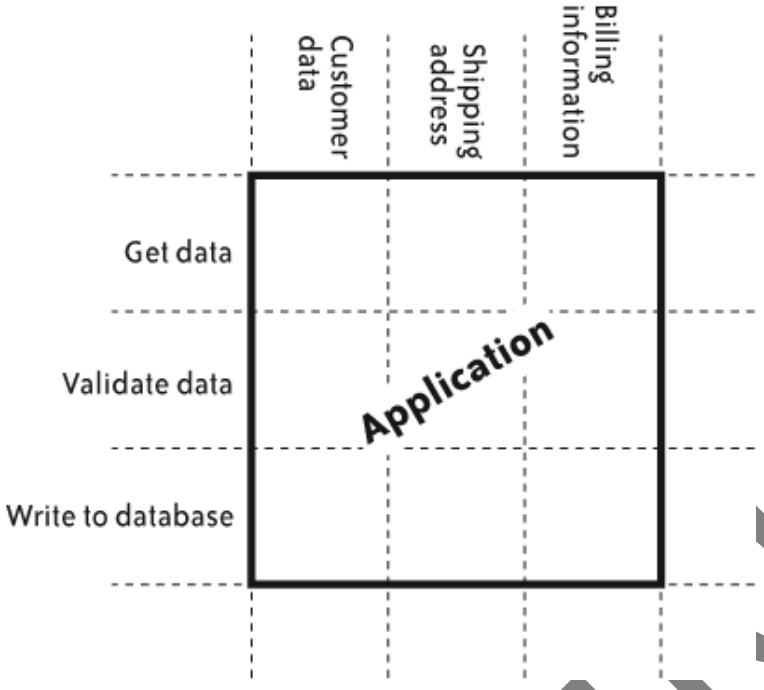
Applying XP to a Brand-New Project (Recommended)

When starting a brand-new XP project, expect the first three or four weeks to be pretty chaotic as everyone gets up to speed. During the first month, on-site customers will be working out the release plan, programmers will be establishing their technical infrastructure, and everyone will be learning how to work together.

Some people think the best way to overcome this chaos is to take a week or two at the beginning of the project to work on planning and technical infrastructure before starting the first iteration. Although there's some merit to this idea, an XP team should plan and build technical infrastructure incrementally and continuously throughout the project as needed.

Starting with a real iteration on the first day helps establish this good habit. Your very first activity is to plan your first iteration. Normally, this involves selecting stories from the release plan, but you won't have a release plan yet.

Instead, think of one feature that will definitely be part of your first release. Brainstorm a few must-have stories for that feature. These first few stories should sketch out a " vertical stripe" (see Figure) of your application.



If the application involves user interaction, create a story to display the initial screen or web page. If it includes reporting, create a story for a report. If it requires installation, create a story for a installer.

These should keep the programmers busy for several iterations. Try to choose stories that the programmers already understand well; this will reduce the amount of time customers need to spend answering programmer questions so they can focus on creating the release plan.

During the iteration, work on just one or two stories at a time and check your progress every day. This will help you deliver completed stories even if your initial plan is wildly inaccurate.

After you've finished planning, programmers should start establishing their technical infrastructure. Set up an integration machine, create your version control repository, and so forth.

Once that's set up, start working on your stories.

During the first iteration, it's a good idea to have all the programmers work on the first few stories as a group. Set up a projector so the whole team navigates while one person drives.

After the first few days, the fundamentals should be well-established and the project should be large enough for people to work on separate parts without unduly interfering with each other.

At this point, you can break into pairs and work normally. It's also a good time to schedule your first coding standards discussion. For that first meeting, you can usually just document what you agreed on while working as a group.

While the programmers are working on stories, customers and testers should work on the vision and release plan and pick a date for your first release.

Each subsequent iteration will be a little easier to plan.

Applying XP to an Existing Project

If you have a legacy project—you can achieve the results, but it will take more time. In this case, adopt XP incrementally.

The big decision

Other than change itself, the biggest challenge in applying XP to an existing project is not writing tests, refactoring, or cleaning up your bug database. The biggest challenge is setting aside enough time to pay down technical debt.

In other words, you incur new technical debt in order to meet your deadlines.

These first steps will allow you to steadily pay down technical debt while continuing to make progress on new stories. As the bug rate for new code drops, you can start organizing your bug backlog.

Organize your backlog

If your team is like most teams, your bug database is full of to-dos, questions, feature requests, and genuine defects. Customers and testers go through the database and eliminate duplicates and unimportant issues. Close feature requests by turning them into stories or rejecting them.

Depending on the size of your bug database, you may not be able to do this work in a single session. Chip away at it every iteration, just as the programmers do with technical debt.

Fix important bugs

Either way, as your bug database becomes a reliable bug repository, make a fix or don't fix decision for each bug. You should probably involve the product manager at some level and you may need the programmers to estimate the cost of fixing some of the bugs.

Close or defer all the bugs that you decide not to fix in this release. You can revisit them when you plan the next release. At this point, all that remains in the database is bugs

that you will fix. Turn these bugs into stories, have the programmers estimate any that remain unestimated, and put them in the release plan.

Move testers forward

When you start this process, your testers will probably spend their time testing each release prior to delivery. A large part of their workload is likely to be manual regression testing.

The programmers' focus on test-driven development will slowly create an automated regression suite and reduce the pressure on the testers.

Emerge from the darkness

This process will allow you to reduce technical debt, increase code quality, and remove defects.

Applying XP in a Phase-Based Organization

XP assumes that you use iterations, not phases, which makes using XP in a phase-based environment difficult. If your organization uses a phase-based approach to development, you may be able to use the XP development practices even if you can't use the other practices.

Your organization may want to try XP within your existing phase-based structure. Your best course of action is to convince your organization to let you try XP's simultaneous phases.

The following suggestions are a starting point; talk to your mentor for more specific advice.

Mandatory planning phase

Your organization may have a planning phase or planning gate that expects you to deliver a detailed plan. If you can, allocate a month for the planning phase and use it to run four actual iterations.

Mandatory analysis phase

If your organization conducts an upfront analysis phase, you may receive a requirements document. In this case, decompose the requirements document into stories. One starting point is to create a story out of each sentence including the words " must," "shall," or "should."

Mandatory design phase

XP assumes the use of incremental design and architecture that is intimately tied to programming with test-driven development. An upfront design phase has little to add to this approach.

If you can, conduct actual XP iterations during the design phase and work on the first stories in your release plan. Use the time to create an initial design and architecture incrementally. Document the results in your design document.

Mandatory coding phase

XP fits well into the coding phase. Break your coding phase into one-week iterations and conduct XP as normal.

Mandatory testing phase

XP performs a lot of testing every iteration. A phase-based organization that considers XP to be the coding phase and expects a long testing phase might schedule too little time for coding and too much time for testing. However, testing is an important part of XP and should remain integrated.

Mandatory deployment phase

With a good build, you should be ready to deploy at the end of any iteration.

You can schedule XP's wrap-up activities for the deployment phase.

Assess Your Agility

Suppose you've been using XP for a few months. How can you tell if you're doing it properly?

The ultimate measure is the success of your project, but you may wish to review and assess your approach to XP as well.

To help achieve this lets do a quiz that focuses on five important aspects of agile development. It explores results rather than specific practices, so you can score well even after customizing XP to your situation.

This quiz assesses typical sources of risk. Your goal should be to achieve the maximum score in each category.

Any score less than the maximum indicates risk, and an opportunity for improvement.

To take the quiz, answer the following questions and enter your scores on a photocopy of the blank radar diagram (Figure 4-2). Don't give partial credit for any question, and if you aren't sure of the answer, give yourself zero points. The result should look something like Figure 4-1.

The score of the lowest spoke identifies your risk, as follows:

- 75 points or less: immediate improvement required (red)
- 75 to 96 points: improvement necessary (yellow)
- 97, 98, or 99: improvement possible (green)
- 100: no further improvement needed

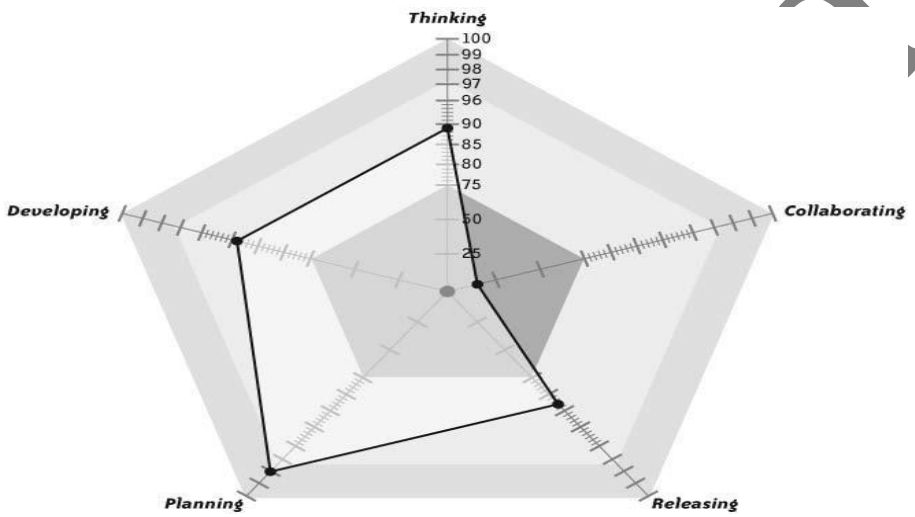


Table 4-1. Thinking

Question	Yes	No	XP Practices
Do programmers critique all production code with at least one other programmer?	5	0	Pair Programming
Do all team members consistently, thoughtfully, and rigorously apply all the practices that the team has agreed to use?	75	0	Pair Programming; Root-Cause Analysis; Retrospectives
Are team members generally focused and engaged at work?	5	0	Energized Work
Are nearly all team members aware of their progress toward meeting team goals?	4	0	Informative Workspace
Do any problems recur more than once per quarter?	0	5	Root-Cause Analysis; Retrospectives
Does the team improve its process in some way at least once per month?	5	0	Retrospectives

Table 4-2. Collaborating

Question	Yes	No	XP Practices
Do programmers ever make guesses rather than getting answers to questions?	0	75	The XP Team

Question	Yes	No	XP Practices
Are programmers usually able to start getting information (as opposed to sending a request and waiting for a response) as soon as they discover their need for it?	4	0	Sit Together
Do team members generally communicate without confusion?	4	0	Sit Together; Ubiquitous Language
Do nearly all team members trust each other?	4	0	The XP Team; Sit Together
Do team members generally know what other team members are working on?	1	0	Stand-Up Meetings
Does the team demonstrate its progress to stakeholders at least once per month?	4	0	Iteration Demo; Reporting
Does the team provide a working installation of its software for stakeholders to try at least once per month?	1	0	Iteration Demo
Are all important stakeholders currently happy with the team's progress?	3	0	Reporting; Iteration Demo; Real Customer Involvement
Do all important stakeholders currently trust the team's ability to deliver?	3	0	Trust; Reporting

Table 4-3. Releasing

Question	Yes	No	XP Practices
Can any programmer on the team currently build and test the software, and get an unambiguous success/fail result, using a single command?	25	0	Ten-Minute Build
Can any programmer on the team currently build a tested, deployable release using a single command?	5	0	Ten-Minute Build
Do all team members use version control for all project-related artifacts that aren't automatically generated?	25	0	Version Control
Can any programmer build and test the software on any development workstation with nothing but a clean check-out from version control?	25	0	Version Control
When a programmer gets the latest code, is he nearly always confident that it will build successfully and pass all its tests?	5	0	Continuous Integration
Do all programmers integrate their work with the main body of code at least once per day?	4	0	Continuous Integration
Does the integration build currently complete in fewer than 10 minutes?	4	0	Ten-Minute Build
Do nearly all programmers share a joint aesthetic for the code?	1	0	Coding Standards
Do programmers usually improve the code when they see opportunities, regardless of who originally wrote it?	4	0	Collective Code Ownership; Refactoring
Are fewer than five bugs per month discovered in the team's finished work?	1	0	No Bugs

Table 4-4. Planning

Question	Yes	No	XP Practices
Do nearly all team members *understand* what they are building, why they're building it, and what stakeholders consider success?	25	0	Vision
Do all important stakeholders *agree* on what the team is building, why, and what the stakeholders jointly consider success?	25	0	Vision

Question	Yes	No	XP Practices
Does the team have a plan for achieving success?	4	0	Release Planning
Does the team regularly seek out new information and use it to improve its plan for success?	2	0	Release Planning
Does the team's plan incorporate the expertise of business people as well as programmers, and do nearly all involved agree the plan is achievable?	3	0	The Planning Game
Are nearly all the line items in the team's plan customer-centric, results-oriented, and order-independent?	4	0	Stories
Does the team compare its progress to the plan at predefined, timeboxed intervals, no longer than one month apart, and revise its plan accordingly?	4	0	Iterations
Does the team make delivery commitments prior to each timeboxed interval, then nearly always deliver on those commitments?	4	0	Iterations; "Done Done"; Slack; Estimating
After a line item in the plan is marked "complete," do team members later perform unexpected additional work, such as bug fixes or release polish, to finish it?	0	25	"Done Done"
Does the team nearly always deliver on its release commitments?	3	0	Risk Management

Table 4-5. Developing

Question	Yes	No	XP Practices
Are programmers nearly always confident that the code they've written recently does what they intended it to?	25	0	Test-Driven Development
Are all programmers comfortable making changes to the code?	25	0	Test-Driven Development
Do programmers have more than one debug session per week that exceeds 10 minutes?	0	3	Test-Driven Development
Do all programmers agree that the code is at least slightly better each week than it was the week before?	25	0	Refactoring; Incremental Design and Architecture
Does the team deliver customer-valued stories every iteration?	3	0	Iterations; Incremental Design and Architecture
Do unexpected design changes require difficult or costly changes to existing code?	0	3	Simple Design
Do programmers use working code to give them information about technical problems?	1	0	Spike Solutions
Do any programmers optimize code without conducting performance tests first?	0	3	Performance Optimization
Do programmers ever spend more than an hour optimizing code without customers' approval?	0	3	Performance Optimization
Are on-site customers rarely surprised by the behavior of the software at the end of an iteration?	4	0	Incremental Requirements
Is there more than one bug per month in the business logic of completed stories?	0	3	Customer Tests
Are any team members unsure about the quality of the software the team is producing?	0	1	Exploratory Testing; Iteration Demo; Real Customer Involvement