Kent beck extreme programming explained pdf

Get the extreme programming explained now with o'Reilly online learning. O'Reilly members experience live online training, as well as books, videos and digital content from more than 200 publishers. Software development projects can be fun, productive and even bold. Yet they can consistently deliver value to a business and remain under control. Extreme Programming (XP) has been designed and developed to meet the specific needs of software development led by small teams in the face of vague and changing requirements. This new, lightweight methodology challenges many conventional principles, including the long-standing assumption that the cost of modifying software necessarily increases dramatically over time. XP recognizes that projects must work to achieve this cost reduction and leverage the savings once they have been earned. XP's core principles include: Distinguishing between decisions to be made by business interests and those that need to be made by project stakeholders. Write unit tests before programming and keep all tests running at all times. Integration and testing of the entire system - several times a day. Production of all software in pairs, two single-screen programmers. Start projects with a simple design that constantly evolves to add the necessary flexibility and eliminate unnecessary complexity. Putting a minimal system into production quickly and developing it in all directions are the most valuable. Why is XP so controversial? Some sacred cows don't cut into XP: Don't force team members to specialize and become analysts, architects, programmers, testers and integrators - every XP programmer participates in all these critical activities every day. Don't conduct complete analysis and design - an XP project begins with a quick analysis of the entire system, and XP programmers continue to make analysis and design decisions throughout development. Develop infrastructure and frameworks as you develop your app, not in advance, enterprise value is the heart rate that drives XP projects. Don't write and maintain implementation documentation - communication in XP projects occurs face-to-face, or through effective testing and carefully written code. You may like XP, or you may hate it, but Extreme Programming Explained will force you to take a fresh look at how you develop software. 0201616416B04062001 Extreme Programming (XP) is an agile software development framework that aims to produce software quality, and a better quality of life for the development team. XP is the most specific of agile frameworks regarding appropriate engineering practices for software development. When this applies, the general features where xp is appropriate have been described by Don Wells on www.extremeprogramming.org: Dynamic change in software requirements Risks caused by fixed-time projects using a new technology Small extended development team co-located The technology you use allows automated unit and functional testing due to specificity when it comes to its comprehensive set of software engineering practices, there are several situations where you may not want to fully practice XP. The post When is XP not appropriate on the C2 Wiki is probably a good place to start finding examples where you may not want to use XP. While you can't use the entire XP framework in many situations, this shouldn't stop you from using as many practices as possible given your context. Values The five values of XP are communication, simplicity, feedback, courage and respect and are described in more detail below. Communication software development is inherently a team sport that relies on communication to transfer knowledge from a team member to all other team members. XP emphasizes the importance of the appropriate type of communication - face-to-face discussion using a whiteboard or other drawing mechanism. Simplicity Simplicity means what is the simplest thing that will work? The purpose of this is to avoid waste and to do only absolutely necessary things such as keeping the system design as simple as possible so that it is easier to maintain, support, and revise. Simplicity also means meeting only the requirements you know; don't try to predict the future. Comments Through constant feedback on past efforts, teams can identify areas for improvement and review their practices. Comments also support simple design. Your team builds something, collects feedback on your design and implementation, and then adjusts your product in the future. Courage Kent Beck defined courage as an effective action in the face of fear (Extreme Programming explained P. 20). This definition shows a preference for action based on other principles so that the results are not detrimental to the team. You need courage to raise organizational issues that reduce the effectiveness of your team. You need courage to stop doing something that doesn't work and try something else. You need courage to accept and act on feedback, even when it is hard to accept. Respect Your team members must respect each other in order to communicate with each other, provide and accept feedback that honors your relationship, and work together to identify simple designs and solutions. Practices The core of XP is the interconnected set of software development practices listed below. Although it is possible to do these practices in isolation, many teams have found that some practices reinforce others and be done together to completely eliminate the risks you often face in software development. XP practices have changed a bit since they were originally introduced. The original twelve practices are listed below. If you want more information about how these practices were originally described, you can visit . The planning game metaphor Simple Design Testing Refactoring Pair Programming Collective Ownership Continuous Continuous 40-hour Standard Standard on-site customer coding below are the practice descriptions described in the second edition of Extreme Programming Explained Embrace Change. These descriptions include improvements based on the experiences of many who practice extreme programming and reflect a more practical set of practices. Sit Together Since communication is one of THE five values of XP, and most people agree that face-to-face conversation is the best form of communication, ask your team to sit together in the same space without communication barriers, such as cabin walls. Entire team A group of interfunctional people with the necessary roles for a product forms a single team. This means that people with a need as well as all people who play a role in satisfaction who need to all work together on a daily basis to achieve a specific outcome. Informative workspace Place your team space to facilitate face-to-face communication, allow people to have some privacy when they need it, and make the team's work transparent to each other and to interested parties outside the team. Use information radiators to actively communicate up-to-date information. Working powered You are most effective in software development and all knowledge work when you are focused and free of distractions. Energetic work means taking steps to ensure that you are physically and mentally able to enter a concentrated state. This means not working too hard yourself (or letting others overwork you). It also means staying healthy and showing respect to your teammates to keep them healthy. Pair programming means that all production software is developed by two people sitting on the same machine. The idea behind this practice is that two brains and four eyes are better than one brain and two eyes. You actually get a continuous review of the code and a quicker response to nagging problems that can stop a dead person in their tracks. Teams that have used pair programming have found that it improves quality and doesn't actually take twice as long because they are able to work through problems faster and they stay more focused on the task at hand, thus creating less code to accomplish the same thing. Stories Describe what the product should do in meaningful terms for customers and users. These stories are meant to be short descriptions of the things that users want to be able to do with the product that can be used for the and serve as reminders for more detailed conversations when the team manages to realize this particular story. Weekly Cycle The weekly cycle is synonymous with iteration. In the case of XP, the team meets on the first day of the week to reflect on the progress made to date, the client chooses the stories they would like to deliver that week, and the team determines how they will approach these stories. The goal by the end of the week is to have tested features that make the selected stories. The intention behind the box delivery time is to produce to show the customer for comments. Quarterly Cycle The quarterly cycle is synonymous with a version. The aim is to maintain the detailed work of each weekly cycle in the context of the overall project. The client presents the team's overall plan in terms of desired features within a particular quarter, which provides the team with a view of the forest while they are in the trees, and it also helps the client to work with other stakeholders who may need an idea of when the features will be available. Remember that when you plan a quarterly cycle, information about a particular story is at a relatively high level, the order of delivery of the story in a quarterly cycle may change, and the stories included in the quarterly cycle may change. If you are able to review the plan on a weekly basis after each weekly cycle, you can keep everyone informed as soon as these changes become apparent to keep the surprises to a minimum. Slack The idea behind slack in XP terms is to add a few low-priority tasks or stories to your weekly and quarterly cycles that can be dropped if the team is behind on more important tasks or stories. In other words, explain the inherent variability of the estimates to ensure you have a good chance of meeting your forecasts. Ten minutes To build The goal with the ten-minute construction is to automatically build the entire system and run all the tests in ten minutes. The founders of XP have suggested a 10-minute delay because if a team has a build that takes longer than that, it is less likely to be run on a frequent basis, thus introducing more time between errors. This practice encourages your team to automate your generation process so that you are more likely to do so on a regular basis and use this automated construction process to run all your tests. This practice supports the practice of continuous integration and is supported by the practice of Test First Development. Continuous continuous integration is a practice where code changes are immediately tested when added to a larger code base. The advantage of this practice is that you can catch and solve integration problems earlier. Most teams fear the code integration stage because of the inherent discovery of conflicts and the resulting problems. Most teams take the If it hurts, avoid it for as long as possible approach. XP practitioners suggest if it hurts, do it more often. The reasoning behind this approach is that if you run into problems every time you code, and it takes a while to find where the problems are, maybe you should integrate more often so that if there are problems, they are much easier to find because there are fewer changes incorporated into the construction. This practice requires some additional discipline and is highly dependent on Ten Minute Build and Test First Development. Test-First Programming Instead of following the normal trajectory of: developing code - writing tests - run tests The practice of Test-First programming follows the path of: from: Automated Test Failure - Run the Failed Test - develop code to pass the test - repeat As with continuous integration, test-First Programming reduces the feedback cycle for developers to identify and solve problems, thus reducing the number of bugs that are introduced into production. Incremental design The practice of incremental design suggests that you do a little work in advance to understand the right width-wise perspective of the system design, and then delve into the details of a particular aspect of that design when you provide specific features. This approach reduces the cost of changes and allows you to make design decisions when necessary based on the most up-to-date information available. The practice of refactoring was originally one of the 12 basic elements, but it was incorporated into the practice of incremental design. Refactoring is an excellent practice to use to keep the design simple, and one of the most recommended uses of refactoring is to eliminate duplication of processes. Roles Although extreme programming specifies specific practices for your team to follow, it doesn't really establish specific roles for the people on your team. Depending on the source you read, there is no guide, or there is a description of how the roles usually found in more traditional projects behave on extreme programming projects. Here are four most common roles associated with extreme programming: The Customer Role The Role of the Client is responsible for making all business decisions regarding the project, including: What should the system do (What features are included and what do they accomplish)? How do we know when the system is being done (what are our acceptance criteria)? How much do we have to spend (what is the funding available, what is the business case)? What do we do next (in what order do we deliver these features)? It is expected that the XP client will be actively involved in the project and ideally become part of the team. The XP client is supposed to be one person, but experience has shown that a person cannot adequately provide all company-related information about a project. Your team needs to make sure that you get a complete picture of the business perspective, but have some ways to deal with conflicts in this information so that you can get a clear direction. The developer Because XP doesn't need much role definition, everyone on the team (with the exception of the client and a couple of secondary roles listed below) is labeled a developer. Developers responsible for the realization of the stories identified by the Client. Because different projects require a different mix of skills, and because the XP method relies on an interfunctional team providing the appropriate mix of skills, the creators of XP didn't feel they needed an additional role definition. The Tracker Some teams may have a tracker as part of their team. It's often one of the developers who spends part of their time every fulfill this additional role. The main objective of this role is to follow the relevant measures that the team deems necessary to monitor their progress and identify areas for improvement. Key indicators your team can track include speed, reasons for gear changes, amount of overtime worked, and pass and fail tests. This is not a role required for your team, and is usually only established if your team determines a real need to keep track of several actions. Coach If your team is just starting to apply XP, you may find it helpful to include a coach in your team. This is usually an external consultant or someone from elsewhere in your organization who has already used XP and is included in your team to help mentor other team members on XP practices and to help your team maintain your self-discipline. The main value of the coach is that they have gone through this before and can help your team avoid the mistakes that most new teams make. Life cycle To describe XP in terms of life cycle, it is probably more appropriate to review the concept of the weekly cycle and the quarterly cycle. First, start by describing the desired results of the project by asking clients to define a set of stories. As these stories are created, the team estimates the size of each story. This size estimate, as well as the relative benefit as estimated by the customer, can provide an indication of the relative value that the customer can use to determine the priority of the stories. If the team identifies certain stories that they are unable to estimate because they do not understand all the technical considerations involved, they can introduce a spike to make targeted research on that particular story or a common aspect of several stories. The time frames are short and time-framed for research on a particular aspect of the project. The spikes may occur before the start of regular iterations or next to the current iterations. Then the whole team comes together to create a release plan that everyone thinks is reasonable. This exit plan is a first pass to find out which stories will be delivered in a particular quarter, or released. The stories delivered must be based on the value they place and on the considerations of how the different stories support each other. Then the team embarks on a series of weekly cycles. At the beginning of each weekly cycle, the team (including the client) meets to decide which stories will be made during this week. The team then divides these stories into tasks to be completed during this week. At the of the week, the team and client review progress to date and the client can decide whether the project should continue or whether sufficient value has been delivered. Delivered. Delivered.