Extreme Programming Explained
Embrace Change

Kent Beck with Cynthia Andres

What is XP?

- XP is about:
  - social change.
  - letting go of old habits and patterns that limit us.
  - giving up defenses that protect us but interfere with our productivity
  - being open about what we are capable of doing.
  - allowing others to do the same.
  - the process of becoming more of our best selves
  - the process of becoming our best as programmers.
- Good relationships lead to good business.
  - Productivity and confidence are related to our human relationships in the workplace as well as our coding ability
  - XP addresses both

What is XP?

- Prepare for success.
- Don’t protect yourself from success by holding back.
- Do your best and then deal with the consequences.
- That’s extreme – you leave yourself exposed.
- This can be as scary as it is exciting and liberating.

- XP focuses on:
  - excellent application of programming techniques
  - clear communication
  - teamwork

What is XP?

- XP Includes:
  - A philosophy of software development based on the values of communication, feedback, simplicity, courage, and respect.
  - A body of practices proven useful in improving software development
    - The practices complement each other, amplifying their effects.
    - The practices are chosen as expressions of the values.
  - A set of complementary principles, intellectual techniques for translating the values into practice
  - A community that shares these values and practices.

What is XP?

- XP is a path to improvement to excellence for people coming together to develop software.
- It is distinguished from other software engineering methodologies by:
  - Short development cycles, resulting in early, concrete, and continuing feedback.
  - An incremental planning approach, which quickly comes up with an overall plan that is expected to evolve over time.
  - The ability to flexibly schedule the implementation of functionality responding to changing business needs
  - Its reliance on automated tests written by programmers, customers, and testers to catch defects early
  - Its reliance on oral communications, tests, and source code to communicate system structure and intent.
  - Its reliance on evolutionary design
  - Its reliance on close collaboration of actively engaged individuals with ordinary talent.
  - Its reliance on practices that work with both short-term instincts of the team members and long-term interests of the project.

What is XP?

- XP is lightweight – you only do what you need to do to create value for the customer.
- XP is a methodology based on addressing constraints in software development.
- XP does not address project portfolio management, financial justification of projects, operations, marketing, or sales.
  - XP has implications in these areas, but does not address them directly.
- XP can work with teams of any size. The practices need to be augmented and altered when many people are involved.
- XP adapts to vague or rapidly changing requirements. XP shines in this area compared to other techniques.

What is XP?

- XP addresses risks at all levels of the development process
  - Schedule slips
    - Short release cycles limit the scope of a schedule slip.
    - Within each release XP uses 1 week iterations of customer requested features yielding continuous feedback.
    - XP plans short tasks for each iteration which aids in problem solving.
    - XP prioritizes the features so that features that slip past the release will be of lower value.

What is XP?

- XP addresses risks at all levels of the development process
  - Project canceled
    - XP asks the business-oriented part of the team to choose the smallest release that makes the most business sense.
    - The value to the company of the software is always maximized.
    - Frequent releases minimize the time spent on developing software that does not fit the needs of the stakeholders.

What is XP?

- XP addresses risks at all levels of the development process
  - System goes sour
    - XP creates and maintains a comprehensive suite of automated tests.
    - The tests are run many times a day to ensure a quality baseline.
    - XP always keeps the system in a deployable condition
    - Problems are not allowed to accumulate

What is XP?

- XP addresses risks at all levels of the development process
  - Defect rate
    - XP tests from the perspective of programmers writing tests function by function.
    - XP tests from the perspective of customers writing tests feature by feature.
  - Business misunderstood
    - XP calls for business-oriented people to be first-class members of the team.
    - The specification of the project is continually refined, so learning by the customer and the team is reflected in the software.

What is XP?

- XP addresses risks at all levels of the development process
  - Business changes
    - With a short release cycle there is less change during the development of a single release.
    - During a release the customer is welcome to substitute new functionality for functionality not yet completed.
  - False feature rich
    - XP insists that only the highest priority tasks are addressed.

What is XP?

- XP addresses risks at all levels of the development process
  - Staff turnover
    - XP asks programmers to accept responsibility for estimating the time required to complete their own work
    - XP gives then feedback about the actual time taken so their estimates can improve.
    - XP respects those estimates.
    - There is less chance for a programmer to get frustrated by being asked to do the impossible.
    - XP encourages human contact among the team reducing loneliness that leads to job dissatisfaction.
    - XP incorporates an explicit model of staff turnover.
    - New team members are encouraged to gradually accept more responsibility, and are assisted along the way by each other and by existing programmers.

What is XP?

- In general terms:
  - XP is giving up old ineffective technical and social habits in favor of new ones that work.
  - XP is fully appreciating yourself for the total effort today.
  - XP is striving to do better tomorrow.
  - XP is evaluating yourself by your contribution to the team's shared goals.
  - XP is asking to get some of your human needs met through software development.

Exploring XP – Values, Principles, and Practices

- Practices are things you do day-to-day
- Values are the large scale criteria we use to judge what we see, think, and do.
- Values bring purpose to practices
- Practices are evidence of values
- Practices are clear
  - Attending the morning standup meeting is a practice
  - Whether I really value communication is fuzzy
- Practices bring accountability to values

Exploring XP – Values, Principles, and Practices

- Values are universal
  - My values at work are exactly the same as my values in the rest of my life
- Practices are dependent on the situation
  - If I want feedback about my programming, continuously building and testing my software makes sense
  - If I want feedback about changing a diaper, continuously building and testing makes no sense.

Exploring XP – Values, Principles, and Practices

- Bridging the gap between values and practices are principles
- Principles are domain specific guidelines for life.
- A gardener might know to plant marigolds next to strawberries
  - A good gardener understands the principles of companion planting.
  - Marigolds naturally repel some of the bugs that eat strawberries.
- We present here the values, principles, and practices of XP
- Becoming a master of XP programming takes time
  - It takes participating in the community of people who share these values and practices.
  - It takes sharing what you know with others.

Values

- Everyone who develops software has a sense of what matters.
  - One person might think what really matters is carefully thinking through all conceivable design decisions before implementing.
  - Another might think what really matters is not having any restrictions on his own personal freedom.
- Much of what people know about software development is focused on individual action.
  - What actually matters is not how any given person behaves
  - What actually matters is how the individuals behave as part of a team or organization.
- Example: Coding style
  - While some styles are better than others,
  - The most important issue is that the team chooses a common style.
  - Idiosyncratic coding styles and the value of individual freedom revealed by the style don't help the team succeed.

Values

- What values should a team focus on to guide development?
  - Communication
  - Simplicity
  - Feedback
  - Courage
  - Respect

Values - Communication

- When problems arise in development, most often someone already knows the solution.
- That knowledge may not get through to someone with the power to make the change.
- Sometimes problems are caused by a lack of knowledge rather than a lack of communication.
- Once you encounter a surprising problem, communication can help you solve it.
  - You can listen to people who have had similar problems in the past.
  - You can talk as a team about how to make sure the problem doesn't recur.
- When you encounter a problem, ask yourselves if the problem was caused by a lack of communication.
  - What communication do you need now to address the problem?
  - What communication do you need to keep yourself out of this trouble in the future?

Values - Simplicity

- To make a system simple enough to gracefully solve only today's problem is hard work.
- Yesterday's simple solution may be fine today, or it may look simplistic or complex.
- When you need to change to regain simplicity, you must find a way from where you are to where you want to be.
- Ask yourself, "What is the simplest thing that could possibly work?"
- Remember, Einstein said, "Everything Should Be Made As Simple As Possible, But Not Simpler."
- Simplicity only makes sense in context.
  - If we need to write a parser with a team that understands parser-generators, a parser-generator is simple.
  - If the team doesn't know about parsing and language is simple, a recursive descent parser is simpler.
- The values balance and support each other.
  - Communication helps achieve simplicity by eliminating unneeded or deferrable requirements.
  - Achieving simplicity gives you that much less to communicate about.

Values – Feedback

- No fixed direction remains valid for long.
- Directions set in advance of experience have a short half-life.
- Change is inevitable.
- Change creates the need for feedback.
- One might to tempted to think that it would be easier to “do it right in the first place.”
  - We may not know how to do it right.
  - There may be several solutions that might work, or no clear solution at all.
  - What’s right today may be wrong tomorrow.
  - Doing everything right today might take so long that changing circumstances tomorrow invalidate today’s solution before it is finished.

Values - Feedback

- Being satisfied with improvement rather than expecting instant perfection, we use feedback to get closer to our goals.
- Forms of feedback:
  - Opinions about an idea, yours or your teammates'
  - How the code looks when you implement the idea
  - Whether the tests were easy to write
  - Whether the tests run
  - How the idea works once it has been deployed
- XP teams strive to generate as much feedback as they can handle as quickly as possible.
  - They try to shorten the feedback cycle to minutes or hours rather than weeks or months.

Values - Feedback

- It is possible to get too much feedback.
  - The team may have to slow down to address the underlying issues that are causing the excess of feedback.
  - Take the time to figure out why there are so many defects and why they take so long to fix.
  - Extend the release cycle period until the feedback issue is resolved.
- Feedback is a critical part of communication.
  - Is performance going to be a problem? I don't know, let's write a little performance prototype and see.
  - Feedback can contribute to simplicity. Which of these three solutions will turn out to be simplest? Try all three and see.
  - While this may seem wasteful, deciding on a non-optimal solution and proceeding with development can lead to much more wasted time later.
  - It is easier to get feedback about a simpler system.

Values - Courage

- Courage is effective action in the face of fear.
- Sometimes courage manifests as a bias to action.
  - If you know what the problem is, do something about it.
- Sometimes courage manifests as patience.
  - If you know there is a problem but you don’t know what it is, it takes courage to wait for the real problem to emerge distinctly.
- Courage as a primary value without counterbalancing values is dangerous.
  - Doing something without regard for the consequences is not effective teamwork.
  - Encourage teamwork by looking to the other values for guidance on what to do when afraid.
- Courage, in concert with other values is powerful.
  - The courage to speak truths, pleasant or unpleasant, fosters communication and trust.
  - The courage to disregard failing solutions and seek now ones encourages simplicity.
  - The courage to seek real concrete answers creates feedback.

Values - Respect

- The value of respect lies below the surface of the other four.
- If members of a team don’t’ care about each other and what they are doing, XP won’t work
- If members of a team don’t care about the a project, nothing can save it.
- For software development to simultaneously improve in humanity and productivity, the contributions of each person on the team need to be respected.

Values – Other

- What is most important is aligning team behavior to team values.
- Other important values include:
  - Safety
  - Security
  - Predictability
  - Quality-of-life
- Holding these values as a team would shape your practices in different ways than the XP values do.
- Values don’t provide concrete advice about what to do in software development.
- Principles bridge the gap between values and practices.

XP Principles

- The principles of XP provide a set of domain-specific guidelines for finding practices in harmony with XP's values.

- Humanity

  - Basic Safety – Freedom from hunger, physical harm, fear of job loss
  - Accomplishment – The opportunity to contribute to society
  - Belonging – The opportunity to identify with a group From which they receive validation and accountability and contribute to its goals
  - Growth – The opportunity to expand their skills and perspective.
  - Intimacy – The ability to understand and be understood deeply by others.

XP Principles

- **Economics**
  - **A technical success may be a hollow victory.**
  - **A real success meets business value, business goals, business needs.**
  - **Two aspects of economics that affect software development:**
    - **The time value of money**
      - **A dollar today is worth more than a dollar tomorrow.**
      - **A software project is more valuable when it earns money sooner and spends money later.**
      - **Incremental design defers investment until the last responsible moment.**
      - **Pay per use realizes revenue from features as soon as they are deployed.**
    - **The option value of systems and teams**
      - **It is valuable to be able to redeploy a media scheduling program for a variety of scheduling-related tasks.**
      - **All the practices are intended to enhance the option value of the software and the team while keeping in mind the time value of money by not investing in speculative flexibility.**

XP Principles

- Mutual Benefit
  - Every activity should benefit all concerned
  - This is a difficult principle to adhere to.
  - There are always solutions to any problem that cost one person while benefitting another.
  - They are a net loss because they create ill will and tear down valued relationships.
  - Extensive internal documentation of software is an example of a practice that violates mutual benefit.
    - You are communicating with some unknown person in the future who might possibly find it helpful. There is no benefit now.
  - XP solves the communication-with-the-future problem in mutually beneficial ways:
    - I write automated tests that help me design and implement better today.
    - I leave the tests for future programmers
    - I carefully refactor to remove accidental complexity, giving me satisfaction and fewer defects and making the code easier to understand for the future.
    - I choose names from a coherent and explicit set of metaphors which speeds development and makes the code clearer.

XP Principles

- **Self Similarity**
  - **In nature we see self similarity at different scales – Think of the seashore or leaves and trees.**
  - **In software development, try copying the structure of one solution into a new context, even at different scales.**
  - **Example: The basic rhythm of development – Write a test that fails, and make it work. This works at other scales:**
  - **In a quarter, list the themes you want to address and address them with stories. Write system level tests expressing the stories and make them work.**
  - **In a week, list the stories you want to address, write tests expressing the stories, and make them work.**
  - **In a few hours, list the tests you know you need to write, write a test, make it work. Write another test, and make them both work until the list in done.**
  - **Having system level tests simplifies design, reduces stress, and improves feedback.**

XP Principles

- **Improvement**
  - **In software development “perfect” is a verb, not an adjective.**
  - **There is no perfect design, no perfect story, no perfect programmer.**
  - **You can, however, perfect your process, your design and your stories.**
  - **Do the best you can today, striving for the understanding necessary to do better tomorrow.**
  - **It doesn’t mean waiting for perfection in order to begin.**
  - **The key to improvement lies in using newfound technological efficiency to enable new, more effective social relationships.**

XP Principles

- Diversity
  - Teams need to bring together a variety of skills, attitudes, and perspectives.
  - Diversity enables us to see problems and pitfalls, to think of multiple ways to solve problems,
  - Two ideas about a design provide an opportunity not a problem.

XP Principles

- Reflection
  - Good teams think about how they are working and why they are working.
  - They don’t try to hide their mistakes, but expose them and learn from them.
  - No one stumbles into excellence.
  - Quarterly and weekly cycles provide time for team reflection, as do pair programming and continuous integration.
  - Reflection can occur in a conversation with a friend or other non-software related reading and activities.
  - Shared meals and coffee breaks provide an informal setting for shared reflection.
  - Reflection can be taken too far. Reflection should come after action. Learning is action reflected.

XP Principles

- **Flow**
  - **Flow means delivering a steady flow of valuable software by engaging in all the activities of development simultaneously.**
  - **The practices of XP are biased towards a continuous flow of activities rather than discrete phases.**
  - **For improvement, deploy smaller increments of value ever more frequently.**
  - **The daily build is flow oriented.**
  - **It is not enough that the software compile and link every day, it should also function correctly every day, or better yet, several times a day.**

XP Principles

- **Opportunity**
  - **Learn to see problems as opportunities for change.**
  - **Problems need to turn into opportunities for learning and improvement, not just survival.**
  - **Can’t make accurate long-term plans? Have a quarterly cycle during which you refine your long-term plans.**
  - **A person alone makes too many mistakes? Program in pairs.**
  - **As you practice XP you will encounter problems at all levels. XP requires that you transform each problem into an opportunity for personal growth, better relationships, improved software.**

XP Principles

- **Redundancy**
  - **The critical, difficult problems in software development should be solved several different ways**
  - **If one solution fails utterly, the other solution will prevent disaster.**
  - **The cost of redundancy is more than paid for by the savings from not having the disaster.**
  - **Example: Defects are a critical difficult problem.**
  - **Defects are addressed by many practices:**
    - **Pair programming**
    - **Continuous integration**
    - **Sitting together**
    - **Real customer involvement**
    - **Daily deployments.**
    - **Some of these practices are redundant, catching some of the same defects.**

XP Principles

- **Failure**
  - **If you're having trouble succeeding, fail.**
  - **Is failure waste? Not if it imparts knowledge.**
  - **If you have several different solutions and don't know which is best, you can:**
    - **Sit around and talk about it**
    - **In the same time, develop several solutions, and find one that works.**
  - **Failure is not to be feared or hidden.**
  - **There is value in failure.**

XP Principles

- Quality
  - Sacrificing quality is not an effective means of control.
  - Quality is not a control variable.
  - Projects don’t go faster by accepting lower quality.
  - They don’t go slower by demanding higher quality.
  - Pushing quality higher often results in faster delivery.
  - We should consider quality with respect to defects, design quality, and the experience of development.
  - Each increase in quality leads to improvement in productivity and effectiveness.
  - People need to do work they are proud of.
  - If you can’t control projects by controlling quality, how can you control them?
    - Time and cost are generally fixed.
    - XP chooses scope as the primary means of planning, tracking, and steering projects.
  - A concern for quality is not an excuse for inaction.
  - Do the best you can today. If you find a cleaner way to do it, you can clean it up later. Improvement is a verb!

XP Principles

- Baby Steps
  - Many small rapid steps are preferable to risky big steps.
  - The overhead of small steps is less than wastefully recoiling from aborted big changes.
  - Baby steps are expressed in practices like
    - Test first programming
    - Continuous integration
- Accepted Responsibility
  - Responsibility can not be assigned, it can only be accepted.
  - Whoever signs up to do work also estimates it.
  - The person responsible for implementing a story is ultimately responsible for the design, implementation, and testing of the story.
  - With responsibility comes authority.
  - When a process expert can tell me how to work but doesn't share in that work, authority and responsibility are misaligned.
  - Neither of us is in a position to use the feedback we need to improve.
  - There is an emotional cost of living with misalignment.

XP Practices

- Practices by themselves are barren.
- Practices are given purpose by values.
  - Pairing to please your boss is just frustrating
  - Pairing to communicate, get feedback, simplify the system, catch errors, and bolster courage makes sense
  - The following practices work well, and work even better together.

XP Primary Practices

- **Sit together**
  - **Develop in an open space big enough for the whole team.**
  - **Meet the need for privacy and "owned space" by having small private spaces nearby.**

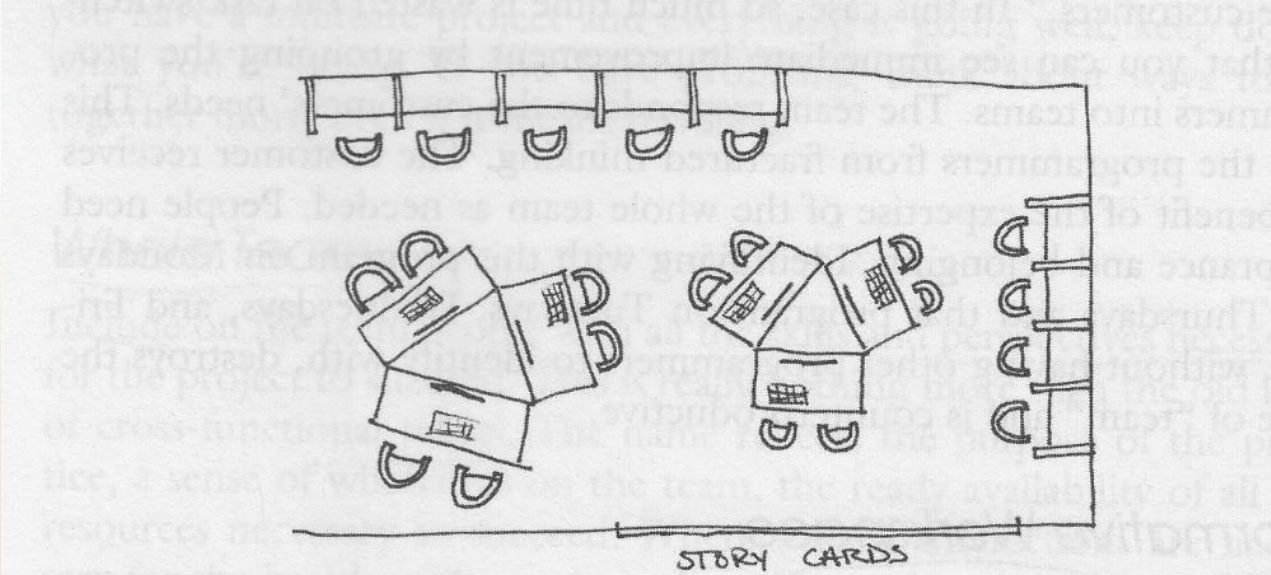

FIGURE 5. A team workspace

XP Primary Practices

- **Whole Team**
  - **Include on the team people with all the skills and perspectives necessary for the project to succeed.**
  - **Ready availability of all the resources necessary to succeed.**
  - **People need a sense of "team."**
    - **We belong**
    - **We are in this together**
    - **We support each others' work, growth, and learning.**
  - **Team size:**
  - **Malcolm Gladwell, "The Tipping Point" describes two discontinuities in team size: 12 and 150. It may be time to split teams when they reach these thresholds.**
    - **12 is the number of people who can comfortably interact in a day.**
    - **150 is about the maximum number of faces on the team you can recognize.**

XP Primary Practices

- **Informative Workspace**
  - **Make your workspace about your work**
  - **Lots of whiteboards for brainstorming sessions**
  - **Putting story cards on a wall**

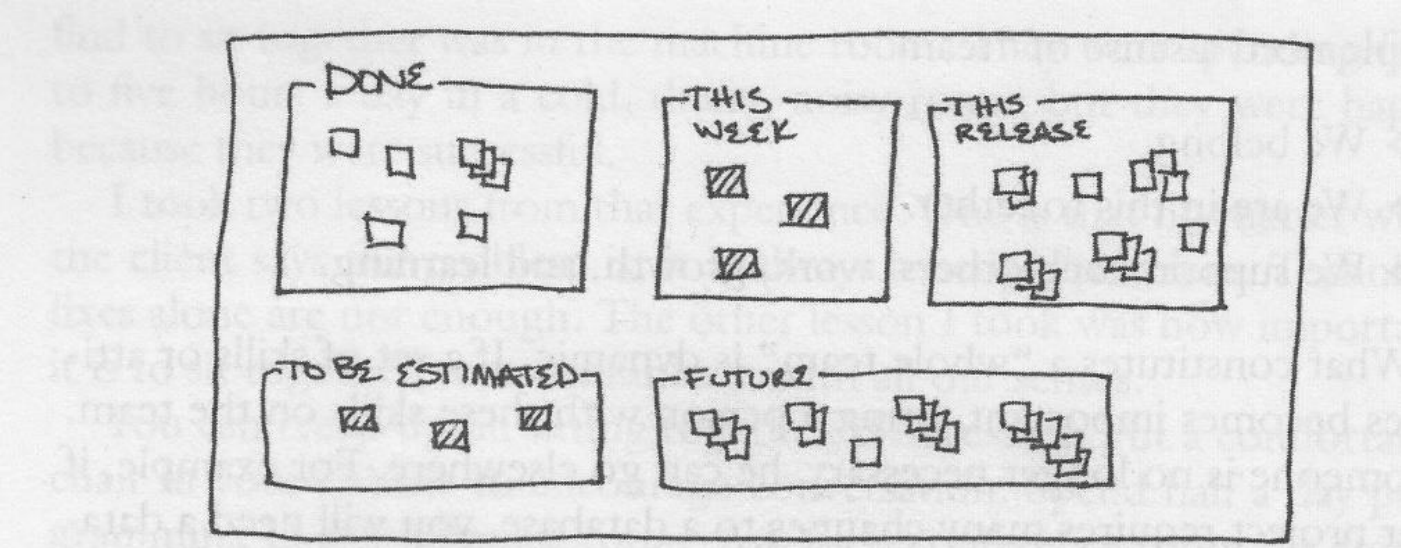

FIGURE 4. Stories on a wall

XP Primary Practices

- **Energized work**
  - **Software development is a game of insight.**
  - **Insight comes to the prepared, rested, relaxed mind.**
  - **Overly long work hours lead to reduced efficiency and can even remove value from a project.**

XP Primary Practices

- Pair Programming
  - Pair programmers:
    - Keep each other on task
    - Brainstorm refinements to the system
    - Clarity ideas
    - Take initiative when their partner is stuck.
    - Hold each other accountable to the team's practices
  - People need both companionship and privacy
    - If you need to work on an idea alone, go doit.
    - Then come back and check in with your team.
    - Come back with an idea if possible rather than code.
  - Take breaks
  - Rotate pairs frequently
  - Pairing and personal space
    - Different individuals and cultures are comfortable with different amounts of body space.
    - Southern Europeans seem to be comfortable at closer quarters than northern Europeans in general
    - Be aware of personal space when pairing with the opposite gender.
    - Personal hygiene, colognes, etc. count.

XP Primary Practices

- Stories
  - Plan using units of customer-visible functionality.
    - “Handle five times the traffic with the same response time.”
    - “Provide a two-click way for users to dial frequently used numbers.”
  - As soon as a story is written try to estimate the time needed to implement it.
  - Estimation give the business and technical perspectives a chance to interact.
  - Early estimation helps get the greatest return from the smallest investment.
  - Give stories short names in addition to a short prose description.
  - Write stories on index cards

XP Primary Practices

- Weekly Cycle
  - Plan work a week at a time
  - Have a weekly meeting on Mondays
  - During the meeting:
    - Review progress to date including a comparison of expected progress vs. actual progress.
    - Have the customers pick a weeks worth of stories to implement
    - Break the stories into tasks
    - Team members sign up for tasks and estimate them
  - Start the week by writing tests for the stories.
  - Spend the rest of the week completing the stories.
  - The nice thing about a weekly cycle is that it is done by Friday.
  - If you find by Wednesday that you are behind schedule, you can choose the most valuable stories and complete them.

XP Primary Practices

- Quarterly Cycle
  - Plan work a quarter at a time.
  - Once a quarter reflect on the team, the project, its progress, and its alignment with larger goals.
  - During the quarterly meeting:
    - Identify bottlenecks, especially those controlled outside the team.
    - Initiate repairs.
    - Plan the theme or themes for the quarter.
    - Pick a quarter's worth of stories to address those themes.
    - Focus on the big picture, where the project fits within the organization.
- Slack
  - Include some minor tasks that can be dropped if you get behind.
  - You can always add more stories later if you have extra time.

XP Primary Practices

- **Ten-minute build**
  - **Automatically build the whole system and run all of the tests in ten minutes.**
  - **A build that takes longer than ten minutes will be used much less often, missing the opportunity for feedback.**
  - **If it takes longer, find ways to optimize the task.**
- **Continuous integration**
  - **Integrate and test changes after no more than a couple of hours.**
  - **The integration step can easily take more time than the original programming.**
  - **The longer you wait the more it costs and the more unpredictable it gets.**

XP Primary Practices

- **Test-First Programming**
  - **Test-first strategy addresses many problems:**
    - **Scope creep – It's easy to get carried away by "just in case" programming. The tests help keep you focused.**
    - **Coupling and Cohesion – If it's hard to write a test, you may have a design problem, not a testing problem. Loosely coupled cohesive code is easy to test.**
    - **Trust – It's hard to trust the author of code that doesn't work. By writing clean, tested code, you give your teammates a reason to trust you.**
    - **Rhythm – It's easy to get lost for hours when you are coding. Test-first makes it clear what to do next: either write another test or make the broken test work. This develops into a natural rhythm: test, code, refactor, test, code, refactor.**

XP Primary Practices

- **Incremental Design**
  - **Invest in the design of the system every day.**
  - **Your design improves as your understanding of the project improves.**
  - **The most effective time to design is in the light of experience.**
  - **Refactoring a design becomes less problematic and stressful as you continue to apply it.**

XP Primary Practices

- Getting Started
  - You are already developing software, You have already started.
  - XP is a way to improve your development process
  - You start where you are and adapt by adding practices that meet your goals and values.
  - As you add practices you find synergies that add more value than you expected.
  - XP works best when it is done altogether, but you need a starting place.
  - What should you do first?
    - Write stories about improving your software development process
      - “Automate the build”
      - “Test first all day”
      - “Pair program with Joe for two hours”
    - Estimate how long each story will take.
    - Pick a story to work on first

XP Primary Practices

- Getting Started
  - Change begins with awareness.
  - Metrics can lead to awareness.
  - Once you are aware of the need for change, you can begin.
  - Primary practices are possible places to begin improving your practices.
  - Pick a practice whose purpose matches you own priorities for change.
  - Change always starts at home. The only person you can actually change is yourself.
    - Programmers can start writing tests first.
    - Testers can automate their tests
    - Customers can write stories and set priorities
    - Executives can expect transparency.
  - Dictating practices to a team destroys trust and creates resentment.

XP Corollary Practices

- **Real Customer Involvement**
  - **Customer involvement leads to programs that are more usable by the stakeholders.**
  - **Customer involvement leads to buy-in. The best program is worthless if the people who are intended to use it prefer the old way.**
- **Incremental Deployment**
  - **When replacing a legacy system with a new one try to avoid a one shot switch to the new system.**
  - **It may lead to unexpected problems and lots of stress.**
  - **Find a little piece of functionality or a limited data set that you can handle right away. Deploy it.**
  - **Run both programs in parallel for a while.**

XP Corollary Practices

- Team Continuity
  - Keep effective teams together.
  - The relationships that have developed take time and history.
- Root Cause Analysis
  - When a defect is found, eliminate the defect and its cause.
  - The idea is not to make the same mistake twice.
  - The XP process for responding to a defect:
    - Write an automated system-level test that demonstrates the defect.
    - Write a unit test with the smallest possible scope that reproduces the defect.
    - Fix the system so the unit test passes. The system test should also pass. If not, go back to step 2.
    - Once the defect is fixed figure out why the defect was created and how to prevent it in the future.

XP Corollary Practices

- **Shared Code**
  - **We need a sense of collective responsibility.**
  - **Anyone on the team can omprove any part of the system at any time.**
  - **“Nobody owns the code.”**
- **Single Code Base**
  - **There is only one code stream**
  - **You can develop temporary branches, but don’t let it live longer than a few hours.**
  - **Fixing one branch of a multiple code base system leads to inconsistent code being worked on by the team.**

XP Corollary Practices

- **Daily Deployment**
  - **Daily deployment is a goal. Rapid Deployment is a step in the right direction**
  - **Programmers should be working with the current deployment to be in synch with each other.**
  - **For daily deployment to work:**
    - **The defect rate must be very low**
    - **Build environment must be automated**
    - **Deployment tools must be automated, including roll back for failures**
    - **Trust in the team and with customers must be high**

XP Corollary Practices

- **Negotiated Scope Contract**
  - **Write contracts for software development that fix time, costs, and quality, but call for ongoing negotiation of the precise scope of the system.**
  - **Reduce risk by signing a sequence of short contracts instead of one long one.**
  - **You can move in the direction of negotiated scope.**
    - **Big, long contracts can be split in half or thirds, with the optional part to be exercised only of both parties agree.**

End