Extreme Programming

John T. Bell

Department of Computer Science
University of Illinois, Chicago

Prepared for CS 442, Spring 2017

Sources

1. Wikipedia: Extreme Programming
2. Wikipedia: Extreme Programming Practices
3. Wikipedia: Kent Beck
4. Kent Beck and Cynthia Andres, "Extreme Programming Explained: Embrace Change", 2nd Edition
5. Kent Beck and Martin Fowler, "Planning Extreme Programming"

An Early Definition of XP

"XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements."

Beck, Kent; Andres, Cynthia. Extreme Programming Explained: Embrace Change (Kindle Location 316). Pearson Education. Kindle Edition. Quoted from the First Edition.

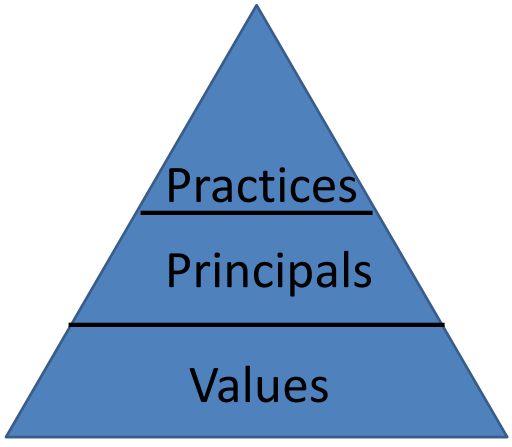
History of XP

- Extreme programming was created by Kent Beck in the late 1990s during his work on the Chrysler C3 payroll project.
- OO movement takes hold; Dot-com boom speeds up expected time-to-market for software.
- XP takes "best practices" to extreme levels. For Example:
  - Frequent inspections -> Pair Programming
  - Test early -> Automated tests built before code.

Kent Beck

- An original signatory to the Agile Manifesto.
- Leading proponent of test-driven development.
- Pioneered Design Patterns and application of Smalltalk.
- Wrote JUnit, with Erich Gamma.
- Popularized CRC cards.
- Works at Facebook.

XP is based upon Values, Principals, and Practices



XP Values

- Communication
- Simplicity
- Feedback
- Courage
- Respect
- Others

Communication Value

- "What matters most in team software development is communication. When problems arise in development, most often someone already knows the solution; but that knowledge doesn't get through to someone with the power to make the change."
- This value drives information sharing and transparency in the XP methodology.

Simplicity Value

- "What is the simplest thing that could possibly work?"
- Start with the simplest possible solution; Add complexity only if necessary.
- XP does not prepare for future possibilities that may or may not happen. It concentrates on the simplest solution for today's problems.

Feedback Value

- Feedback from the system: Automated unit tests and continuous integration return feedback on code changes within minutes.
- Feedback from the customer: Frequent close contact, including acceptance tests, planning meetings, and general project steering.
- Feedback from the team: When requirements change, team gives new planning estimates.

Courage Value

- "Courage is effective action in the face of fear."
- "Sometimes courage manifests as a bias to action. Sometimes courage manifests as patience."
- "Courage alone is dangerous; In concert with the other values it is powerful. The courage to speak truths, pleasant or unpleasant, fosters communication and trust. The courage to discard failing solutions and seek new ones encourages simplicity. The courage to seek real, concrete answers creates feedback."

Respect Value

- Respect for team mates.
- Respect for the project.
- Respect for the customer.
- Respect for self.

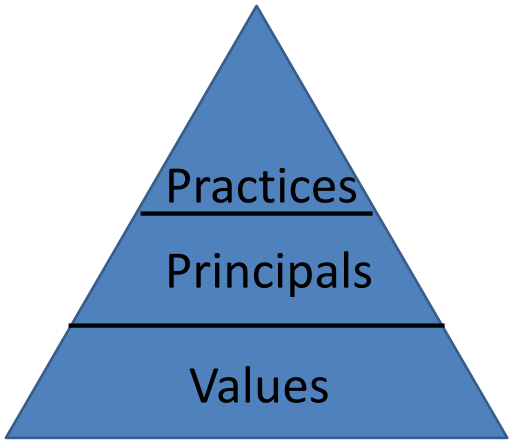
Other Values

- The values given above are those that come with XP.
- Any given organization, team, or project may and should add their own values.
- Some possibilities include safety, security, predictability, and quality-of-life.

Poll: Which XP Value do you see as most important?

A. Communication
B. Simplicity
C. Feedback
D. Courage
E. Respect

XP is based upon Values, Principals, and Practices



XP Principals bridge the gap between values and practices

- Humanity
- Economics
- Mutual Benefit
- Self-Similarity
- Improvement
- Diversity
- Reflection
- Flow
- Opportunity
- Redundancy
- Failure
- Quality
- Baby Steps
- Accepted Responsibility

Principals I

- Humanity – People develop software.
- Economics – Add business value.
- Mutual Benefit – Strive for win-win always.
- Self-Similarity – If something works in one situation, try to apply it to others.
- Improvement – "Perfect" is a verb, not an adjective. Always strive to improve processes.

Principals II

- Diversity – It takes a variety of skills and perspectives to solve problems effectively. This can lead to conflicts when there are multiple possible solutions or approaches.
- Reflection – ***How*** and ***why*** does this work?
- Flow – Bias towards a continuous flow of development, as opposed to developing in phases or chunks. Continuous integration.

Principals III

- Opportunity – "Learn to see problems as opportunities for change."
- Redundancy – "The critical, difficult problems in software development should be solved several different ways." e.g. defect elimination.
- Failure – Trial and error. Try things even if they don't work, and learn from the failures.
  (Edison learned 1000s of filaments that failed.)

Principals IV

- Quality – Quality is not a control variable. Always strive for high quality, and control the project by adjusting scope as needed.
- Baby Steps – Lots of little steps can be faster than a few large bounds, with more control.
- **Accepted** Responsibility – Responsibility for completing tasks must be taken, not given.

Coming Soon

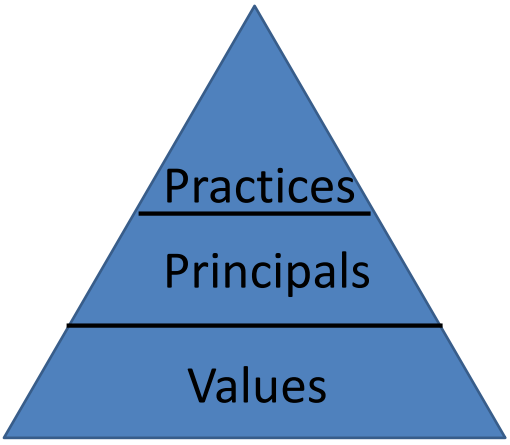
- Next we will look at the specific **Practices** of XP.

Exercise: Reflection

Based on your past experience in SE, e.g. 440:

- What have you done that worked well, that you would do again on future projects?
- What have you done that has not worked well, and what could you change to (try to) make it work better in the future?

XP is based upon Values, Principals, and Practices



XP Practices – Primary and Corollary

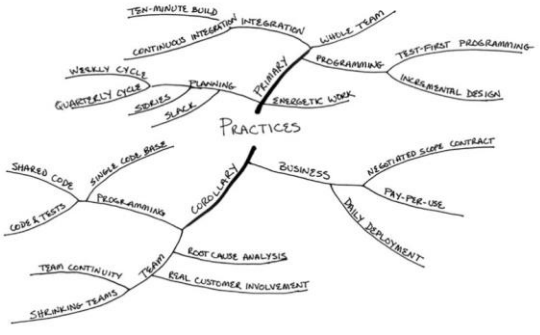


FIGURE 3. Summary of practices

Beck, Kent; Andres, Cynthia. Extreme Programming Explained: Embrace Change (Kindle Location 788). Pearson Education. Kindle Edition.

- Primary practices safely give immediate benefits.
- Corollary practices should only be attempted after mastering primary practices.
- Combining practices amplifies their effectiveness.

The List of Primary Practices

- Sit Together
- Whole Team
- Informative Workspace
- Energized Work
- Pair Programming
- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- Ten-Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

Primary Practices I

- Sit Together – Find a large room where everyone can sit together, at least part of the day.
- Whole Team – Bring together all skills and perspectives necessary. Foster sense of team.
- Informative Workspace – The space should be a visible display of the project and its current status. The space should also have resources for positive social interactions, e.g. coffee & snacks.

Informative Workspace Images

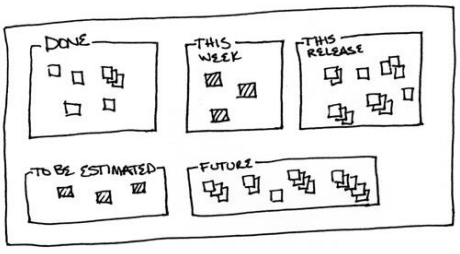


FIGURE 4. Stories on a wall

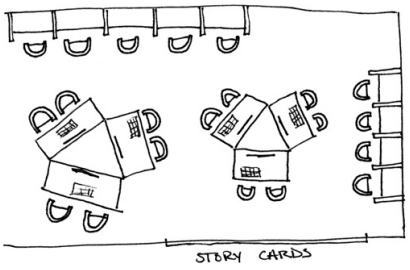


FIGURE 5. A team workspace

Beck, Kent; Andres, Cynthia. Extreme Programming Explained: Embrace Change (Kindle Location 845). Pearson Education. Kindle Edition.

Primary Practices II

- Energized Work – Work only as many hours as you are productive and efficient.
- Pair Programming – Two people, One computer.
  - Keep each other on task.
  - Brainstorm refinements.
  - Clarify ideas.
  - Alternate initiative when one is stuck.
  - Hold each other accountable to team standards.
  - Rotate pairs after 1 to 3 hours. No more than 5 or 6 hours a day.
  - Beware of inter-personal, hygiene, and other social issues.

Primary Practices III

- Stories – A more natural alternative to "requirements". Functionality described by the client and quickly estimated by the development team.

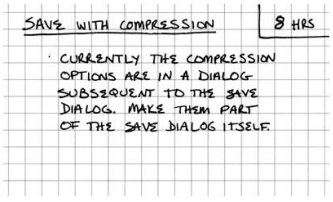


FIGURE 7. Sample story card

Beck, Kent; Andres, Cynthia. Extreme Programming Explained: Embrace Change (Kindle Location 845). Pearson Education. Kindle Edition.

Primary Practices IV

- Weekly Cycle – Plan each week at a Monday meeting:
  1. Review progress to date, including how actual progress for the previous week matched expected progress.
  2. Have the customers pick a week's worth of stories to implement this week.
  3. Break the stories into tasks. Team members sign up for tasks and estimate them. (Alt: Draw tasks from hat/pile.)
- Start the week by writing automated tests that will run when the stories are completed. Then spend the rest of the week completing the stories and getting the tests to pass. Deliver functionality & celebrate every Friday.

Primary Practices V

- Quarterly Cycle – Another good time frame for planning longer-term goals. During quarterly planning:
  - 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.

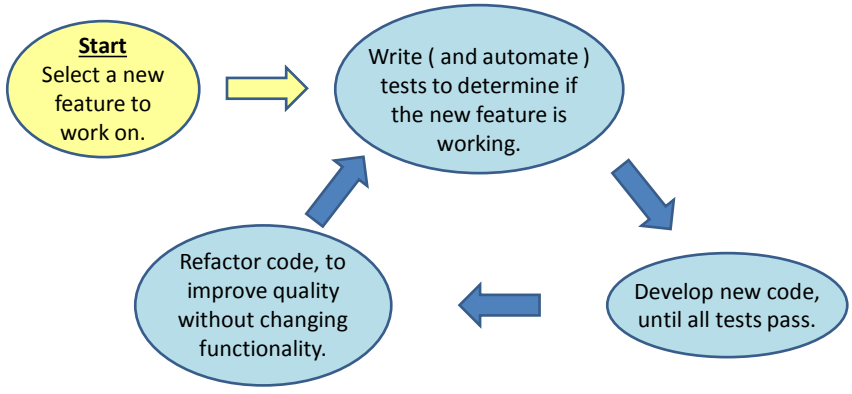
Primary Practices VI

- Slack – Always include some minor tasks that can be dropped if needed to meet overall commitments. Don't over-commit and under-deliver, but set realistic attainable goals.
- Ten-Minute Build - Automatically build the whole system and run all of the tests in ten minutes.

Primary Practices VII

- Continuous Integration - Integrate and test changes after no more than a couple of hours.
- Test-First Programming - Write a failing automated test before changing any code.
  - Avoid scope creep by keeping focused goals.
  - Build trust by writing code that passes tests.
  - Develop a rhythm: test, code, refactor, repeat.
  - Difficulty writing tests indicates a design problem.

Test-First Development Cycle



Primary Practices VIII

- Incremental Design – Make small safe design improvements every day. (Software is easier and cheaper to redesign than bricks and mortar, if done properly.)

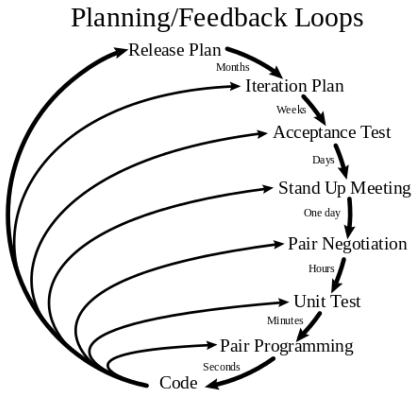
Think-Pair-Share
Which one(s) do you want to try out?

- Sit Together
- Whole Team
- Informative Workspace
- Energized Work
- Pair Programming
- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- Ten-Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

FYI – The Corollary Practices

- Real Customer Involvement
- Incremental Deployment
- Team Continuity
- Shrinking Teams
- Root-Cause Analysis
- Shared Code
- Code andTests
- Single Code Base
- Daily Deployment
- Negotiated Scope Contract
- Pay-Per-Use

Planning Extreme Programming



- This material is presented based on an excerpt from "Planning Extreme Programming" by Kent Beck and Martin Fowler, handed out in class. See "Resources".
- Image source: https://commons.wikimedia.org/wiki/File:XP-feedback.gif , originally authored by Don Wells.