Case Study Retrospective: Kent Beck's XP Versions 1 and 2

Laurie Williams
North Carolina State University
williams@csc.ncsu.edu
 Agenda

• Extreme Programming (XP) 2nd Edition
  • Values
  • Principles
  • Primary Practices
  • Corollary Practices

• Retrospective Look at Empirical Studies of XP1 Teams
  • IBM
  • Sabre
Values, Principles, and Practices

Values

Purpose

Practices

Accountability

Principles
XP2 Values

- **Values** are the roots of things we like and don’t like in a given situation.

- **Communication:** Often when a problem arises, someone knows the solution but knowledge doesn’t get around to person who needs it. [sustained]

- **Simplicity:** Making a bet that it is better to do a simple thing today and pay a little more to change it if it needs it, than to do a more complicated thing today that may never be used. [sustained]

- **Feedback:** No fixed direction stays valid for long. [sustained]

- **Courage:** Effective action in the face of fear. [sustained]

- **Respect:** Team members must care about each other and the project. [new]
XP2 Principles

- Principles are domain-specific guidelines.
- Humanity
- Economics
- Mutual benefit
- Self-similarity
- Improvement
- Diversity
- Reflection
- Flow
- Opportunity
- Redundancy
- Failure
- Quality
- Baby steps
- Accepted responsibility
XP2 Practices: Primary

Practices are the things you do day-to-day.
Sit Together [new]

• Develop in an open space big enough for everyone.
• Have small, private spaces nearby.
Whole Team [new]

• 1st class cross-functional team

• Tipping Points [Malcolm Gladwell]
  • 12: # of people who can comfortably interact in a day
  • 150: above this you no long recognize the faces of everyone on the team
Informative Workspace [new]
Energized Work
[was 40-Hour Week]

• Work only as many hours as you can be productive and only as many hours you can sustain.

• Tired developers make more mistakes, which slows you down more in the long run (remove value from product).

• If you mess with people’s personal lives (by taking it over), in the long run the project will pay the consequences.
Pair Programming [sustained]

• Two software engineers work on one task at one computer

• One engineer, the driver, has control of the keyboard and mouse and creates the implementation

• The other engineer, the navigator, watches the driver’s implementation to identify defects and participates in on-demand brainstorming

• The roles of driver and observer are periodically rotated between the two software engineers
Stories
[wasy Planning Game (User Stories)]

• Customer-visible functionality

---

Edit SR Details Screen

Add "Cancel" button to undo changes and return to previous page.

TC: Check that browser returns to correct previous change.

Estimated: 24
Actual: 
Weekly cycle
[was Planning Game]

- Highest priority stories in “time boxed” weekly increments
- Caveat: see Slack practice
Quarterly Cycle [was Small Releases]

- Timeboxed

- As small as possible, but still delivering business value
  - No releases to ‘implement the database’

- Get customer feedback early and often
Slack [new]

- In every iteration, plan some lower-priority tasks that can be dropped if you get behind – builds trust if you don’t miss the “important stuff.”

Ten-Minute Build [new]

- **Automatically** build the entire system and run all tests in 10 minutes
- Feedback, feedback!
Continuous Integration [sustained]

• Pair writes up unit test cases and code for a task (part of a user story)

• Pair unit tests code to 100%

• Pair integrates

• Pair runs ALL unit test cases to 100%

• Pair moves on to next task with clean slate and clear mind

• Should happen once or twice a day.

• Prevents IntegrationHell [integration could take longer than programming]
Test-first Programming [sustained]

- **Test-Driven Development (TDD)**
  - Write tests before code
  - Tests are automated
  - Often use xUnit framework
  - Must run at 100% before proceeding

- **Acceptance Testing**
  - Written with the customer
  - Acts as “contract”
  - Measure of progress
Incremental Design
[was Simple Design and Refactoring]

• No Big Design Up Front (BDUF)

• Knowledge-based design – the most effective design is in light of experience

• “Do The Simplest Thing That Could Possibly Work”
• “You Aren’t Gonna Need It” (YAGNI)

• Refactoring: Improve the design of existing code without changing functionality
  • Simplify code
  • Opportunity for abstraction
  • Remove duplicate code

• Relies on testing to ensure nothing breaks in the process of refactoring
# XP2 Primary Practice Summary

<table>
<thead>
<tr>
<th>XP2 Primary Practice</th>
<th>Sustained/New/ XP1 Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sit together</td>
<td>New</td>
</tr>
<tr>
<td>Whole team</td>
<td>New</td>
</tr>
<tr>
<td>Informative workspace</td>
<td>New</td>
</tr>
<tr>
<td>Energized work</td>
<td>40-hour week</td>
</tr>
<tr>
<td>Pair programming</td>
<td>Sustained</td>
</tr>
<tr>
<td>Stories</td>
<td>Planning game</td>
</tr>
<tr>
<td>Weekly cycle</td>
<td>Planning game</td>
</tr>
<tr>
<td>Quarterly cycle</td>
<td>Small releases</td>
</tr>
<tr>
<td>Slack</td>
<td>New</td>
</tr>
<tr>
<td>Ten-minute build</td>
<td>New</td>
</tr>
<tr>
<td>Continuous integration</td>
<td>Sustained</td>
</tr>
<tr>
<td>Test-first Programming</td>
<td>Testing</td>
</tr>
<tr>
<td>Incremental Design</td>
<td>Simple Design Refactoring</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>XP1 Practice</th>
<th>Disposition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metaphor</td>
<td>Removed</td>
</tr>
<tr>
<td>Collective code ownership</td>
<td>Corollary: Shared code</td>
</tr>
<tr>
<td>On-site customer</td>
<td>Corollary: Real customer involvement</td>
</tr>
<tr>
<td>Coding standard</td>
<td>Removed</td>
</tr>
</tbody>
</table>
XP2 Practices: Corollary

Corollary Practices

• **Real Customer Involvement [was On-Site Customer].** Customer available on site to clarify stories and to make critical business decisions.

• **Incremental Deployment [new].** Gradually deploy functionality. Big deployment is high risk and can have high human and economic costs.

• **Team Continuity [new].** Keep effective teams together.

• ** Shrinking Team [new].** As a team grows in capability, keep the workload constant but gradually reduce the size (e.g. with attrition).
**Corollary Practices (cont’d)**

- **Root-Cause Analysis [new]**.  (1) write failing automatic system test; (2) write failing automatic unit test; (3) get each to pass; (4) examine how defect was created and not caught

- **Shared Code [was Collective Code Ownership]**. Anyone on the team can improve any part of the system at any time. [prereq: pair programming, continuous integration; test-first programming]

- **Code & Tests [was Simple Design]**. Maintain only the code and tests as permanent artifacts. Rely on social mechanisms to keep alive the important history of the project.

- **Single Code Base [new]**. Have only one code stream.
Corollary Practices (cont’d)

- **Daily Deployment [new]**. Put new code into production every night.

- **Negotiated Scope Contract [new]**. Fix time, cost, and quality but call for on-going negotiation of precise scope.

- **Pay-per-use [new]**. Charge for every time the system is used.
Extreme Programming Examination

### Extreme Programming Evaluation Framework

XP-EF (said X-pef)

- **XP-Context Factors (XP-cf)**
- **XP-Adherence Metrics (XP-am) (said X-pam)**
- **XP-Outcome Measures (XP-om) (said X-pom)**

- Reusable framework for reporting:
  - the extent to which an organization has adopted XP practices; and
  - the result of this adoption
IBM: XP-Context Factors (XP-cf)

- Small team (7-10)
- Co-located
- Web development (toolkit)
- Supplier and customer distributed (US and overseas)

- Examined one release “old” (low XP) to the next “new” (more XP)
# IBM: XP-Outcome Measures (XP-om)

<table>
<thead>
<tr>
<th>XP Outcome Measures</th>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response to Customer Change (Ratio (user stories in + out) /total)</td>
<td>NA</td>
<td>0.23</td>
</tr>
<tr>
<td>Pre-release Quality (test defects/KLOEC of code)</td>
<td>1.0</td>
<td>0.50</td>
</tr>
<tr>
<td>Post-release Quality (released defects/KLOEC of code)</td>
<td>1.0</td>
<td>0.61</td>
</tr>
<tr>
<td>Productivity (stories / PM)</td>
<td>1.0</td>
<td>1.34</td>
</tr>
<tr>
<td>Relative KLOEC / PM</td>
<td>1.0</td>
<td>1.70</td>
</tr>
<tr>
<td>Putnam Product. Parameter</td>
<td>1.0</td>
<td>1.92</td>
</tr>
<tr>
<td>Customer Satisfaction</td>
<td>NA</td>
<td>High  (qualitative)</td>
</tr>
<tr>
<td>Morale (via survey)</td>
<td>1.0</td>
<td>1.11</td>
</tr>
</tbody>
</table>
Sabre: XP-Context Factors (XP-cf)

- Small team (6-10)
- Co-located
- Scriptable GUI environment
- Customer remote, multinational, several time zones

- Examined third release “old” (low XP) to the ninth release “new” (sustained XP)
## Sabre: XP-Outcome Measures (XP-om)

<table>
<thead>
<tr>
<th>XP Outcome Measures</th>
<th>Old</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response to Customer Change (Ratio (user stories in + out) /total)</td>
<td>NA</td>
<td>N/A</td>
</tr>
<tr>
<td>Pre-release Quality (test defects/KLOEC of code)</td>
<td>1.0</td>
<td>0.25</td>
</tr>
<tr>
<td>Post-release Quality (released defects/KLOEC of code)</td>
<td>1.0</td>
<td>0.70</td>
</tr>
<tr>
<td>Productivity (stories / PM) Relative KLOEC / PM Putnam Product. Parameter</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>1.46</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2.89</td>
</tr>
<tr>
<td>Customer Satisfaction</td>
<td>NA</td>
<td>High (anecdotal)</td>
</tr>
<tr>
<td>Morale (via survey)</td>
<td>N/A</td>
<td>68.1%</td>
</tr>
</tbody>
</table>
## Summary

Two characteristically-agile teams:

When used by teams operating within the specified context, the use of a specified subset of XP practices leads to an improvement in . . .

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>IBM Case study evidence?</th>
<th>Sabre case study evidence?</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-release quality</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>post-release quality</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>programmer productivity</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>customer satisfaction</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>team morale</td>
<td>Yes</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## XP2 Primary Practices

<table>
<thead>
<tr>
<th>XP2 Primary Practice</th>
<th>IBM</th>
<th>SABRE-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sit together</td>
<td>No – adjoining cubes</td>
<td>Yes</td>
</tr>
<tr>
<td>Whole team</td>
<td>No</td>
<td>Yes (customer rep sitting at times)</td>
</tr>
<tr>
<td>Informative workspace</td>
<td>No</td>
<td>Yes (many big visible charts)</td>
</tr>
<tr>
<td>Energized work</td>
<td>Sustainable pace</td>
<td>Sustainable pace</td>
</tr>
<tr>
<td>Pair programming</td>
<td>50% anecdotal</td>
<td>50% anecdotal</td>
</tr>
<tr>
<td>Stories</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Weekly cycle</td>
<td>Yes</td>
<td>10 day</td>
</tr>
<tr>
<td>Quarterly cycle</td>
<td>5 months</td>
<td>Yes</td>
</tr>
<tr>
<td>Slack</td>
<td>Not likely</td>
<td>Not likely</td>
</tr>
<tr>
<td>Ten-minute build</td>
<td>No</td>
<td>No (hours to build)</td>
</tr>
<tr>
<td>Continuous integration</td>
<td>Nightly or more</td>
<td>Daily</td>
</tr>
<tr>
<td>Test-first Programming</td>
<td>Progress in unit testing</td>
<td>Progress in unit testing</td>
</tr>
<tr>
<td></td>
<td>No automated acceptance testing</td>
<td>Some automated acceptance testing</td>
</tr>
<tr>
<td>Incremental Design</td>
<td>SDUF</td>
<td>No design doc</td>
</tr>
<tr>
<td></td>
<td>Limited refactoring</td>
<td>Limited refactoring</td>
</tr>
</tbody>
</table>

© Laurie Williams 2005
# XP1 Primary Practice Rejects

<table>
<thead>
<tr>
<th>XP1 Practice</th>
<th>IBM</th>
<th>SABRE-A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metaphor</td>
<td>No</td>
<td>System of names</td>
</tr>
<tr>
<td>Collective code ownership</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>On-site customer</td>
<td>No (remote, responsive to email)</td>
<td>On-site marketing rep (1/2 time; email)</td>
</tr>
<tr>
<td>Coding standard</td>
<td>Yes</td>
<td>Naming standard</td>
</tr>
</tbody>
</table>

© Laurie Williams 2005
Conclusions

- XP2 has 13 primary practices
  - Can do individually, work best together
- XP2 has 11 corollary practices
  - Best to start using these once have a core set of primary practices
- XP2 seems more “reasonable” than XP1
- Two small, co-located, successful XP1 teams were studied
  - IBM team used:
    » ~8 of the 13 XP2 primary practices
    » 2 of 4 XP1 rejected XP1 primary practices
  - Sabre team used:
    » ~12 of 13 XP2 primary practices
    » ~3 of 4 XP1 rejected XP1 primary practices