XP Practices . . . or Best Practices?

An examination of the XP practices

Laurie Williams
North Carolina State University
The “13” Practices of XP

1. Stand-up Meeting
2. On-site Customer
3. Sustainable Pace
4. Continuous Integration
5. Small Releases
6. Coding Standards
7. Pair Programming
8. Test-Driven Development
9. Refactoring
11. Collective Code Ownership
12. Simple Design
13. Metaphor (as a substitute for architecture)
“The 13th Practice”
The Stand Up Meeting
On-Site Customer

• If you can do it, why not?
• Better yet, on-site end-user

<table>
<thead>
<tr>
<th>WHY PROJECTS FAIL</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Incomplete Requirements</td>
<td>13.1</td>
</tr>
<tr>
<td>2. Lack of user involvement</td>
<td>12.4</td>
</tr>
<tr>
<td>3. Lack of resources</td>
<td>10.6</td>
</tr>
<tr>
<td>4. Unrealistic Expectations</td>
<td>9.9</td>
</tr>
<tr>
<td>5. Lack of executive support</td>
<td>9.3</td>
</tr>
<tr>
<td>6. Changing requirements</td>
<td>8.7</td>
</tr>
<tr>
<td>7. Lack of planning</td>
<td>8.1</td>
</tr>
<tr>
<td>8. Didn’t need it any longer</td>
<td>7.5</td>
</tr>
<tr>
<td>9. Lack of IT management</td>
<td>6.2</td>
</tr>
<tr>
<td>10. Technology illiteracy</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Sustainable Pace
Continuous Integration

“Treat the daily build as the heartbeat of the project.”

-- Jim McCarthy, Dynamics of Software Development

• It minimizes integration risk
• It reduces the risk of low quality.
• It supports easier defect diagnosis.
• It improves morale.

Small Releases

• Get customer feedback early and often
• “An iterative approach is required that allows an increasing understanding of the problem through successive refinements, and to incrementally grow an effective solution . . .”
• Focus on delivering business value
Coding Standards

- Use Coding Conventions
- Write *intention-revealing code, rather than comments*
Pair Programming (1)

- **Pre-jelling**
  - 60-80% increase in cost for development
  - Increased quality
  - 2-80 hours to jell

- **Post-jelling**
  - 0-15% increase in cost for development
  - Increased quality
  - Anecdotes abound
    » Almost always indicate cost savings
  - Positive economic picture
Pair Programming (2)

Expected benefits

- Higher product quality
- Improved cycle time
- Increased programmer satisfaction
- Enhanced learning
- Ease staff training and transition
- Knowledge management
- Reduced product risk
- Enhanced team building

Some concerns

- Can everyone be more effective this way?
- What about personality conflicts?
- The transition is tough
- Need for co-location
- More research is necessary to validate findings on larger scale
Test-Driven Development (1)

- **NCSU experiments**
  - 3 companies, 24 developers
  - 18% more test cases passed
  - 16% more time (not statistically significant)
  - Good test coverage (98% Method, 92% Statement, 97% Branch)

- **University of Karlsruhe**
  - 19 students
  - No significant difference in time (IP and AP)
  - Test-first less reliable after first implementation (IP)
  - No significant difference (test-first slightly higher) in final reliability of the produced results (AP)

- **IBM – did design**
  - 2500 automated unit tests + 100 automated performance tests run daily
  - 400 interactive tests run rarely
  - Reduced defect density by 50% with no increase in cycle time
Test-Driven Development (2)

- Writing automated unit tests
- 100% unit test passing criteria
- Solicitation/automation of customer acceptance tests
- Long-term benefits of test assets created
- Using TDD as a design technique
- More research needed
Refactor Mercilessly

- Improve the design of existing code without changing functionality
  - Simplify code
  - Opportunity for abstraction
  - Remove duplicate code

- Relies on automated unit testing to ensure nothing breaks in the process of refactoring
Release Planning (a.k.a. The Planning Game)

- Requirements via User Stories
- Dependant on on-site customer
- Priorities chosen/revisited via “The Planning Game” after each increment/release
- Maintains intensity on project goals
Collective Code Ownership

- No wait on time the owner
- Relies heavily on a complete set of unit tests
- Code Ownership organizational pattern
  - Authored by Jim Coplien in 1995
Simple Design

- No time wasted on documenting designs that don’t “pan out” when it comes to the realities of implementation or due to requirements churn
- Reduce time spent on obsolete designs
- How much “rework” does this cause?
- What are the long-term effects of lack of documentation?
Metaphor (as a substitute for architecture)
Seeding the Discussions

• Many of the XP practices can be considered best practices or “near” best practices

• These can prudently be used in:
  • Creating Lean Disciplined Methods
  • Synthesizing Hybrid Agile/Disciplined Methods

• Research is needed (aka Agile/Disciplined Research Issues)
  • “Near” best practices
  • Innovative/revolutionary practices
More Information

- Laurie Williams
- 919-513-4151
- williams@csc.ncsu.edu
- http://collaboration.csc.ncsu.edu/laurie

- Conferences
  - XP2003 (May 2003, in Genova, Italy)
    http://xp2003.org
  - Agile Software Development (June 2003, Utah)
    http://agiledvelopmentconference.com
  - XP/Agile Universe (August 2003, New Orleans)
    http://agileuniverse.com

- Agile Bibliography
  - http://collaboration.csc.ncsu.edu/agile/Bibliography.htm