

An Example Thesis in Software Engineering at CNU

Mike Long

**A Meta-Analysis of the Quality and
Productivity Benefits of Pair Programming
and Test-Driven Design**

Meta-analysis

What

- ❑ Individual studies are often not conclusive
- ❑ Average results across studies and apply sophisticated statistical analysis

Why

- Software managers are likely to accept Extreme Programming on a piecemeal basis
- If some of the practices can be validated they are much more likely to be accepted
- Literature is inconsistent

XP Practices

- Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership

Practices cont.

- Continuous integration
- 40-hour week
- Onsite customer
- Coding standards

Pair Programming

- ❑ Two programmers working side by side
- ❑ Switch roles periodically
- ❑ One programmer at the keyboard actively creating design, production code, test code
- ❑ Other programmer watching, catching defects, providing another perspective

PP Pros and Cons

- Constant review
- Constant training
 - ⇐ Pairs switch among all team members
- Increased problem solving skills?
- Twice the investment in programmer time
- Interaction problems?

Test Driven Design

- Programmers write unit tests before they write their code
- Integrate these test into a larger suite of tests that are run often
- Keep tests running as long as software is in use
- Fuses activities of design and coding

TDD Pros/Cons

- Increases programmer's confidence.
- Objective Boolean answer as to whether or not a program works.
- Concrete documentation about what a program really does.
- Makes software malleable instead of brittle.
- Possible to devise the simplest possible solution because the cost of change is manageable.

Write Tests When

- The interface of a method is unclear – write test first
- The interface is clear but implementation is even slightly complicated – write test first
- Unusual circumstance where method should still work – write test to make explicit
- Bug in method – test to reveal and fix
- Refactoring code with missing tests – write tests before refactoring

Test Driven Design

- ❑ Test driven design results in a significant quality improvement over the control group
- ❑ The effect should be noticeable
- ❑ Extremes
 - ↔ On study show it hurts
 - ↔ One study showed huge improvements

Example study

- ❑ North Carolina State University
- ❑ Both student and professional programmers
- ❑ 16% and 18% quality improvement

Pair Programming

- ❑ **Pair programming results in a moderately large quality improvement over the control group**
- ❑ **Typical study the Nagappan Freshman PP Study**
 - ↔ **Spanned three semesters of an introductory programming course**
 - ↔ **Experimental and control equivalent SAT and experience**
 - ↔ **Assessment score 2.05 for pairs and 1.62 for solo**

TDD Productivity

- ❑ **Least conclusive results**
- ❑ **Barely noticeable productivity DECREASE over control group**
- ❑ **Typical study by Bobby George**
 - ↳ **Professional programmers**
 - ↳ **16% increase in amount of time require to practice test-driven design as compared to control group**

PP Productivity

- ❑ Small sample sized
- ❑ Productivity decrease
- ❑ Typical example Macias/Holcombe XP Study
 - ↔ 20 teams of fourth year undergraduate students doing a real development project in industry.
 - ↔ Effect size of 0.9

Conclusions PP Quality

- ❑ Pair programmers do produce higher quality code
- ❑ Effect size equally large for both student and professionals – useful pedagogical tool
- ❑ Student studies – large improvements on programming assignments but not on tests
- ❑ PP unlike TDD is not difficult to learn
- ❑ Produces equally high quality among all project sizes.

Conclusions PP Productivity

- ❑ Pair programming is significantly less productive over the short term
- ❑ Some indications that pairs perform well with rapidly changing environments or difficult algorithms

TDD Quality

- ❑ Large effect size. Conclusive proof. TDD improves quality
- ❑ Difficult to learn. Probably programmers did not have time to make the paradigm shift from test last to test first.
- ❑ Size of project is important and large projects underrepresented.
 - ⇐ TDD allows changes throughout the software development lifecycle.

Conclusions TDD Productivity

- Effect size slightly below the smallest noticeable effect size.
- TDD same productivity – results less conclusive
- Largest decreases in productivity in industry with professional programmers.
- Takes considerable time to learn short term productivity losses
- Cheaper of the two alternatives for infusing higher quality into software