Pragmatic Software Quality for Small (and Large) Software Companies

Rick Hall, Ph.D.
ChoiceMaker Technologies, Inc.
My background

- Lapsed physicist
- No formal degree in Computer Science
- Worked at 9 different software-related organizations in the past 15 years
  - Solo consulting
  - IT department of Dow Jones & Company
  - Logic Works, Inc. (About 250 employees)
  - Platinum Technologies (10,000 employees)
  - Office.com (200 employees)
  - ChoiceMaker Technologies
ChoiceMaker Technologies

- Record-matching software
- 3 patents
- 4 products
- 8 full-time employees, 2 part-time
- 4 full-time developers, 1 part-time
- About to hire our fifth full-time developer
A Provocative Thesis

- Nobody writes good quality software
A Revised Thesis

- **WRONG**: Nobody writes good quality software
  Counter-example: Mars Rover

- Nobody at a commercial company writes good quality software
A Revised Thesis

- **WRONG:** Nobody writes good quality software

- **WRONG:** Nobody at a commercial company writes good quality software

  Counter-example: SQL Compass at Logic Works, Inc.
A Defensible Thesis

- Successful software companies write as little software as possible, and the software they do write is of minimal quality

  Focus on value-added software
  Stick to (open-source) industry standards where possible
  Borrow or license software wherever possible

  Set “good enough” standards for software quality
A Old Idea

- Schedule/Quality/Scope triangle

Resources are usually not an issue. They’re either available or they’re not. Also adding resources can slow a project down (Frederick Brooks, Mythical Man Month).

Control two of the three parameters; for example, scope and quality, but not schedule

Best practice: quality and schedule, sacrifice scope
A Tale of Three Products, I

- ERwin (Entity-Relation modeling for Windows)
  - Horrendous code
  - Met customers’ needs
  - Extremely profitable
  - Still being sold by Computer Associates
A Tale of Three Products, II

- Universal Directory (Data-warehousing utility)
  - Bad but not totally horrendous code
  - Customers could live without it
  - Never profitable, but not a lot of resources spent on it
  - Withered away a few years after launch
A Tale of Three Products, III

- SQL Compass (would-be successor to ERwin)
  - Innovative frameworks
  - High-quality code
  - Mammoth effort
  - Never made it to market
Moral of the story

- Get a product to market and let customers decide
- Don’t bet huge resources until demand is demonstrated
- High-quality software is not an end in itself
Bad quality is not enough

- Code quality needs to meet certain minimums, but after that it is usually not critical
- Good designs are more important
  - Require a lot of work
  - Hard to get right
  - Multiple iterations are inevitable
- Correct domain analysis is even more important
- Clear, written use cases are the basis for analysis
  - Require a lot of work, but it is done on paper, not in code
  - Hard to get right
  - Multiple iterations are inevitable
Software development is a tribal activity

- Teams
- Roles (Rational Unified Process)
- Ownership
- Smart people who can work together
  - Lots of time spent finding the right people
  - Current candidate search
    - 400 resumes
    - 27 first phone interviews, a dozen second phone interviews
    - Handful of in-person interviews
    - At least 160 hours of work
- Programming requires literacy
  - Explaining complex ideas in code
  - Hard to write good essays, so why should code be different?
  - Good authors are rare, so why should good programmers be common?
Good developers are driven by pride

- Good quality developers like to produce good quality code

- Developers don’t argue with a compiler; similarly, give developers tools to quantify the quality of their code

- Another provocative thesis: code reviews are worse than useless because they are too slow, too subjective, too divisive and can’t be repeated as part of an automated testing system. They also can reduce a developer’s sense of ownership, which is a cardinal sin.
A bit of theory

- Open-Closed principle: a completely stable package is open for extension but closed for change

- John Lakos, Large-scale C++ Construction
  - Reduce physical couplings
  - Eliminate cyclical dependencies
  - Layered software for testing

- Bertrand Meyer, Object-Oriented Software Construction
  - Pre- and post-conditions, invariants
  - Objects created in a valid state and must remain valid
Tools for measuring Java code quality

- Bug counts (traditional)
  - Bugzilla

- JUnit
- JDepend (open-closed principle, cyclic dependencies)
- Checkstyle
- PMD
- Simian (monkey-at-the-keyboard code copying)
- Growing list

- Issue: how to automate all these tools
Maven

- **Build tool**
  - Competes with make, Ant
  - Can incorporate Ant scripts

- **Plug-in framework**

- **Automated testing**
  - JUnit
  - JDepend
  - Checkstyle
  - PMD
Contact Info

Rick Hall
Software Architect
ChoiceMaker Technologies, Inc.
48 Wall Street
11th Floor
New York, NY 10005
(212) 919-4413
rphall@choicemaker.com
www.choicemaker.com