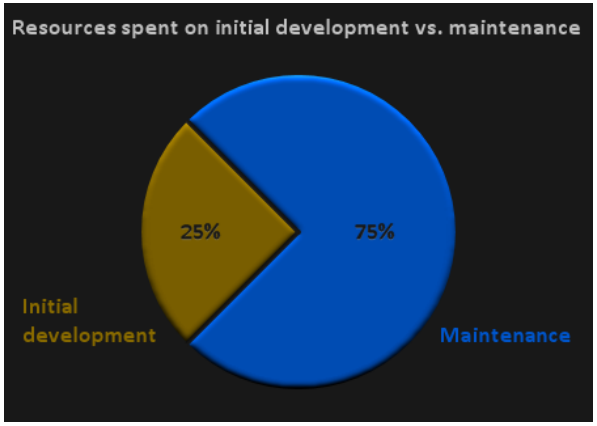
Software Sustainment:

Pay Now or Pay Later



SSTC 2012
Arlene Minkiewicz, Chief Scientist
PRICE Systems, LLC
arlene.minkiewicz@pricesystems.com

TruePlanning® by PRICE® Systems

Optimize tomorrow today.

Agenda

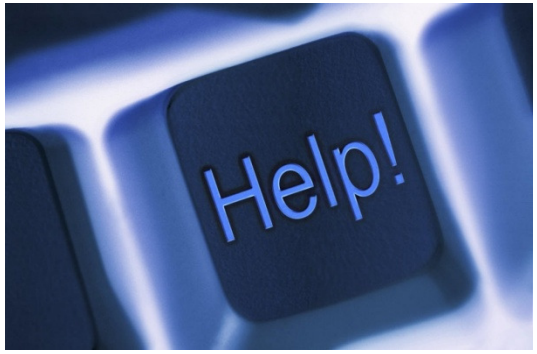
- Introduction
- Software Sustainment and Maintenance
- Quality Software
- Quality Practices
- Conclusions and Recommendations

Introduction

- Budgets are getting tighter – fewer new programs
  - More time and effort on maintenance and sustainment
  - Need to build more maintainable systems going forward
- Research based on a study in progress intended to study overall sustainment costs
- Software structural quality identified as significant driver for software maintenance costs
- Increased software quality decreases software sustainment costs associated with maintenance of the software
- This paper focuses on quality practices and trade-offs between development costs and quality of delivered system

Software Sustainment

- Software sustainment refers to anything that needs to be done to keep a software program delivering its required functionality
- Activities during sustainment include
  - Fixing bugs
  - Adding new features
  - Upgrade for changing environments
  - Address technical debt
  - Help desk
  - Training
  - Operational Support



Software Maintenance

- Subset of software sustainment activities
- Software maintenance is defined as the process of modifying, for update or repair existing operational software while leaving primary function intact. (SWEBOK)
- Activities included in maintenance:
  - Preventive
  - Corrective
  - Perfective
  - Adaptive

Software Quality

- Software quality has two dimensions
  - Functional Quality - Doing the right thing
  - Structural Quality - Doing it the right way
- Consortium for IT Software Quality (CISQ) is developing a standard for structural quality of software based on:
  - Reliability
  - Performance
  - Security
  - Maintenance
  - Size

Structural Quality

- Reliable, efficient software has the following characteristics:
  - Well thought out, easy to understand and well documented architecture
  - Minimized complexity
  - Error and exception handling pervasive
  - Programming best practices applies
  - Sound resource management practices

Structural Quality

- Secure software has the following characteristics:
  - Well thought out, easy to understand and well documented architecture
  - Bug free
    - 90% of software vulnerabilities result of defects (SEI 2005)
    - Security breaches are caused by faulty specifications, designs and implementations
    - Buffer overflows continue to top the list of hacker helpers

Structural Quality

- Maintainable software has the following characteristics:
  - Well written
  - Well documented and comprehensively commented
  - Follows programming best practices
  - Organized in a logical fashion
  - Modular
  - Includes suite of acceptance and regression tests

Quality Best Practices

- Pair programming, peer reviews or inspections
- Test Driven Development
- Continuous integration with automated tests
- Automated static code analysis
- Quality Documentation
- Good Naming conventions

Pair Programming and Peer Reviews

- Pair programming involves two developers working on the code at any given time
  - One computer
  - One driver, one navigator
  - Roles change frequently
- Peer Reviews or Inspections involve a more formal review of development artifacts
  - One or more person reviews other's work
  - Varying levels of formality

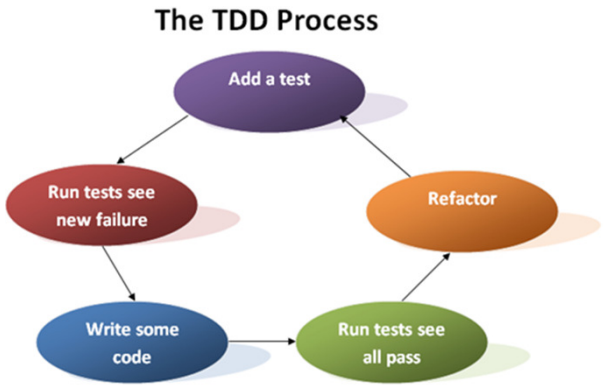
Pair Programming and Peer Reviews

- There is evidence of their value. In several studies the number of development tests that pass increased by
  - 15% , "The Cost and Benefits of Pair Programming", A. Cockburn
  - 14%, "Strengthening the Case for Pair Programming", L. Williams
  - Capers Jones' list pair programming and inspections at the top of his list of best practices for defect removal

- The evidence is mixed on productivity impact
  - Some studies find an increase while others experience decreases.

Test Driven Development (TDD)

- No code is written for a feature until the tests for that feature are written
  - Originally tests fail since no code has been written
  - Just enough code to make it pass
  - Once it passes, refactoring occurs to make it cleaner, simpler, using test to ensure it continues to pass



Test Driven Development (TDD)

- Tests conducted at Microsoft in two different environments showed 2.5x and 4.2x defect rate decreases between projects of similar size and scope, one using TDD and one without
- An article in IEEE Software documents 18 studies across the industry with 10 documenting improved quality with TDD, 7 with inconclusive results and only one with a quality decrease
- The Microsoft study also indicated that there was a slight increase in development time for the projects using TDD

Continuous Integration

- Changes in code base are ‘continuously’ integrated into an operational system
  - Automated process integrated with automated test suite
  - Real time feedback for bad behavior
  - Broken builds become high priority
  - Frequent integrations ease analysis of problem
- No study that specifically declares continuous integration = high quality
  - One expert reports observing projects that use continuous integration have dramatically less bugs in production
  - Steve McConnell suggests that a benefit of frequent builds is reduced risk of low quality
- Continuous integration requires investments in hardware, software and human resources

Automated Static Code Analysis

- Static code analysis focuses on the structural soundness of code
  - Pattern matching, best practices and standards
  - Quality and maintainability metrics
- Evidence supports the effectiveness of static analysis
  - Study by S. Xiao found a 6x defect reduction
    - "Performing high efficiency source code static analysis with intelligent extensions"
  - Capers Jones cites the use of static analysis tools as best practice for defect removal, second only to reviews and inspections
- Static code analysis requires significant investment in software and effort to configure, maintain and analyze the environment

Good Documentation

- Software is easier to fix and change if it can be understood.
- Software with quality documentation – design, architecture, requirements, code – is easier to understand
- According to the SWEBOK, 40-60% maintenance effort is spent determining where to make a change or correction.
- A study in *Empirical Software Engineering* found that software engineers with good documentation spent 21.5% less time understanding the software than those with only source code

Naming Conventions

- Which Line of code would be easier to understand?

```
X = Y * Z;

Square_Area = Height * Width;
```

- While it seems like a simple thing, naming conventions are huge determiners of maintainability
- File structure should also have use sensible naming conventions
- There's no indication that good names take longer to think og than bad ones

Conclusions and Recommendations

- For most software systems more money is spent maintaining than developing
- Quality infused (or not) during development significantly impacts the amount of maintenance required and the efficiency with which it can be maintained and enhanced
- Teams should select best quality practices based on
  - Size of project
  - Size and distribution of team
  - Nature of team and organization
  - Investment required

Questions?

Arlene Minkiewicz, Chief Scientist
PRICE Systems, LLC
arlene.minkiewicz@pricesystems.com