

# Taming the Software Development Process

Tools, Tips, and Techniques

Michael Swanson  
Senior Consultant  
Microsoft Corporation

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Introduction

- Who am I?
- Team sizes
  - The lone developer
  - Small team of 2-5 developers
  - Larger team of 5+ developers
- Techniques discussed are applicable to both Formal and Agile methodologies
  - Methodologies are not mutually exclusive and can be successfully mixed
  - The goal is software, not methodology

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Software Estimation

## Term

- What does Webster say about an *estimate*?
  - to judge tentatively or approximately the value, worth, or significance of
  - to determine roughly the size, extent, or nature of
  - to produce a statement of the approximate cost of
- An estimate is an estimate. It is not necessarily a commitment!

# Software Estimation

## What We Know

- Estimates are difficult to get right
- Early estimates are not as accurate as later estimates
- It is much easier to estimate small tasks
- Depending on who you ask...
  - 30% of software projects are cancelled
  - 50% overrun their schedule and/or budget
  - 20% make their deadlines and budgets

# Software Estimation

## What We Can Do

- Never commit to an estimate you don't believe
  - If forced to estimate without enough information, one million years is generally a safe bet. 😊
- Know what you don't know
  - If you don't understand something, estimate and schedule a *spike*
  - Only build long enough to make an accurate estimate
- Break down large tasks into smaller tasks

# Software Estimation

What We Can Do (continued)

- Estimate everything
  - Design, coding, testing, code reviews, documentation, localization, vacation, etc.
  - Don't forget to plan for time to estimate
- Never let managers tell programmers to reduce an estimate
- Agile approach uses concept of *velocity*
  - *Stories* estimated in arbitrary units, not hours
  - Customer selects stories that fit into next iteration without exceeding velocity



# Software Estimation

## Example

- Estimate the time it would take to perform a task if you already understood its design
- Use industry or historical metrics to compute other components of task based on percentages
- Components will vary based on each project, so think!

	A	B	C	D	E	F	G
1			25%	50%	5%	5%	
2	<b>Task</b>	<b>Write Code</b>	<b>Detailed Design</b>	<b>Write Tests</b>	<b>Code Review</b>	<b>Document</b>	<b>Dev Hours</b>
3	Account data layer	16	4.00	8.00	0.80	0.80	29.60
4	Account business entity	32	8.00	16.00	1.60	1.60	59.20
5	Account creation screen	32	8.00	16.00	1.60	1.60	59.20
6	Account maintenance screen	24	6.00	12.00	1.20	1.20	44.40
7	Loan approval logic	16	4.00	8.00	0.80	0.80	29.60
8							

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Unit Testing

## Kinds of Testing

- Different kinds of testing
  - *Unit* for small sections of code
  - *Integration* to test if your code works well with others
  - *System* for manual or automated testing by a QA team
  - *Stress* to push an application until it fails
  - *Beta* to allow users early access to unfinished code for feedback
  - *Acceptance* to prove that it meets specified requirements

# Unit Testing

Test-Driven Development (TDD)

- Develop a test
- Get it to fail
- Write code to pass the test
- *Refactor*
  - A relatively small transformation that alters the internal structure of the code without changing its external behavior
- Repeat

# Unit Testing

## Tips

- *Test first* is not required, although it is beneficial
- Unit testing is a development activity
- Unit tests should be run frequently
- Tests should be completely independent of each other
- Write tests that exploit found bugs
- Focus only on the unit being tested
- Code coverage is interesting, but not required

# Unit Testing

Tips (continued)

- For data layers, handy to use framework
- Tests can be included in:
  - The original code
  - A separate assembly (DLL)
  - A conditional compile
- Can optionally be distributed
- Ongoing debate about testing private methods
- Mock Objects for missing components or dependencies

*demo*

Unit Testing

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources



# Continuous Integration

## Overview

- Build represents “health” of the project
- Daily build is good, but more frequent builds are better
- Integrate early and often
  - Helps flush out risk earlier in lifecycle while there is more time to respond
- System alerts developers to problems
  - E-mail build failures to entire team
  - E-mail status to developers who check-in

# Continuous Integration

## Overview (continued)

- Can start immediately on a project
- Avoids fragmented development effort by identifying problems early
- Small integration failures are easier to diagnose than large ones
- Agile's *collective ownership* means that anyone can work on any code
- Commonly runs unit tests
  - Unit test failure can optionally be considered a build failure

# Continuous Integration

## Process

- The automated continuous integration server:
  - Performs a full check-out from source control
  - Cleans the build output folder
  - Forces a complete rebuild of the entire project
  - Executes all unit tests (optional)
  - Runs other optional tools (like FxCop, NDoc, etc.)
  - Reports on the build and unit testing status

# Continuous Integration

## Tips

- Create and test build script first
  - NAnt is a popular build tool
  - MSBuild in future
- Perform a full check-out
- Always rebuild the entire solution
- Raise build visibility by using something like the Ambient Orb™

*demo*

Continuous Integration

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Code Reviews

## Overview

- Commonly known, but rarely performed
- Microsoft has found that it takes 3 hours to fix a defect using code inspection versus 12 hours using testing
- Agile proponents often argue that pair programming eliminates the need for formal review
- As studies have shown, the earlier a defect is discovered, the less expensive it is to fix
- Helps to have coding standards

# Code Reviews

## Purpose

- Determine if code has issues that will lead to defects
- Watch for boundary conditions that might not be caught by testing
- Review algorithm selection and performance
- Evaluate future maintainability of code
- Does it follow recommended guidelines?
- Teach guidelines



# Code Reviews

## Process

- Use metrics or feature complexity/risk to select code for review
- Establish requirements for code to be reviewed. Examples:
  - Code must compile
  - Must pass style check
  - Must pass FxCop analysis
- Take a snapshot of the code for review and e-mail it to participants

# Code Reviews

Process (continued)

- Code can be reviewed by individuals or in a group setting
- If in a group, reviewers must understand code intent and read it before the review
  - I prefer to print out a hard copy with line numbers and mark it up
- If a meeting, the author must be present
- Project code and walk through asking for comments from reviewers

# Code Reviews

Process (continued)

- Track agreed-upon changes/suggestions
  - Bug tracking software
  - Follow-up to verify that changes are actually made
- Helpful to log review time against LOC for estimating future meetings
- Benefits increase as team members become used to techniques
  - Better to start at beginning of project than near the end

# Code Reviews

## Things to Check

- Code follows established coding standards
- Good comments and documentation
- Performs correct function (unit tests can help here)
- Appropriate use of APIs
- Poor algorithm choice
- Handling exceptional conditions (errors)
- Correct resource management (Graphics.Dispose)
- Security

# Code Reviews

## Tips

- Expect code to be criticized
- Not a place to rearchitect the solution
  - For large changes, schedule another meeting
- If a reviewer "signs off" on the code, he is just as responsible as the original author
- Unit tests check a lot of functionality. Instead, concentrate on techniques and optimizations.
- If static analysis wasn't used for pre-check, use for post-check (i.e. FxCop)

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Source Code Metrics

## Overview

- Easy way to get a quick snapshot of the entire code base
- Helps to identify problematic areas for code review
- Help to determine velocity of the project
- Typically tracks lines of code (LOC), statements, comments, methods per class, average and maximum depth, complexity, etc.

# Source Code Metrics

## Complexity

- One of the most useful metrics is Tom McCabe's *Cyclomatic Complexity*
- To compute:
  - Start with 1 for the straight path through the routine
  - Add 1 for each of the following keywords or their equivalents: if, while, repeat, for, and, or
  - Add 1 for each case in a case statement



# Source Code Metrics

## Complexity Example

```
while (nextPage != true)
{
    if ((lineCount <= linesPerPage) && (status !=
        Status.Cancelled) && (morePages == true))
    {
        // ...
    }
}
```

- Start with 1 for the routine
- Add 1 for the *while*
- Add 1 for the *if*
- Add 1 for each &&
- Total calculated complexity of 5
- Anything with a greater complexity than 10 or so is an excellent candidate for simplification or refactoring

# Source Code Metrics

## Complexity Benefits

- Easy to compute
- Unlike other complexity measurements, it can be computed immediately in the development lifecycle (which makes it Agile-friendly)
- Provides a good indicator of the ease of future code maintenance
- Can help focus testing efforts
- Makes it easy to find complex code for formal review

*demo*

Source Code Metrics

# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Working with Offshore

## Overview

- Lots of recent discussion about working with offshore teams
- 2001 Aberdeen Group study on average annual IT salaries in:
  - China           \$4,750
  - India           \$5,850
  - Russia         \$7,500
- Overall savings estimates range from 15% (Meta) to 25% (Forrester) to a high of 65% (Aberdeen)

# Working with Offshore

## Challenges

- Communication is a challenge
- Different time zone
  - For example, India is 10.5 hours ahead
  - Prepare to schedule meetings at odd hours
- Different language and culture
  - Meaning can be misinterpreted or lost
- Spend more time writing documents, sending instant messages, sending e-mail, and talking on the phone
- Methodologies may not easily mesh

# Working with Offshore

## Tips

- Doesn't work as well with Agile processes which encourage constant communication
  - Consider small "task sheets"
- For larger projects, consider bringing over an ambassador
  - May cost a little more
  - Improves communication
  - Can answer questions about meetings, changing requirements, specifications, etc.

# Working with Offshore

Tips (continued)

- CMM Level 5 does not necessarily equate to quality code...be sure to monitor/review
- Can likely afford to send work back and still meet cost objectives
  - However, remember that mistakes cost time on both sides
- Select isolated features/functionality
- Automated continuous integration ensures code is healthy for dispersed teams



# Agenda

- Introduction
- Software Estimation
- Unit Testing
- Continuous Integration
- Code Reviews
- Source Code Metrics
- Working with Offshore
- Resources

# Resources

## Books

- *Coder to Developer* by Mike Gunderloy - <http://www.codertodeveloper.com/>
- *Code Complete, Second Edition* by Steve McConnell - <http://www.microsoft.com/MSPress/books/6822.asp>
- *Test-Driven Development in Microsoft .NET* by James W. Newkirk and Alexei A. Vorontsov - <http://www.microsoft.com/MSPress/books/6778.asp>
- *Domain-Driven Design: Tackling Complexity in the Heart of Software* - <http://domaindrivendesign.org/book/>
- *Refactoring: Improving the Design of Existing Code* by Martin Fowler - <http://martinfowler.com/books.html#refactoring>
- *Continuous Integration* by Martin Fowler (online article) - <http://www.martinfowler.com/articles/continuousIntegration.html>
- *Balancing Agility and Discipline* by Barry Boehm and Richard Turner - <http://www.aw-bc.com/catalog/academic/product/0,1144,0321186125,00.html>

# Resources

## Tools

- Task Tracker - <http://www.positive-g.com/tasktracker/>
- NUnit - <http://www.nunit.org/>
- NUnitASP - <http://nunitasp.sourceforge.net/>
- CruiseControl.NET - [www.continuousintegration.net/](http://www.continuousintegration.net/)
- Draco.NET - <http://draconet.sourceforge.net/>
- NAnt - <http://nant.sourceforge.net/>
- FxCop - <http://www.gotdotnet.com/team/fxcop/>
- SourceMonitor - <http://www.campwoodsw.com/>

# Resources

Contact

<http://blogs.msdn.com/mswanson/>

***Microsoft***<sup>®</sup>

© 2004 Microsoft Corporation. All rights reserved.

This presentation is for informational purposes only. Microsoft makes no warranties, express or implied, in this summary.